"KUZUSHIJI

RECOGNITION"

A PROJECT WORK-II REPORT SUBMITTED TO

THE NATIONAL INSTITUTE OF ENGINEERING, MYSURU

(AUTONOMOUS UNDER VTU)

In partial fulfillment of the requirements for project work – II, eight semester

Bachelor of Engineering

in

Computer Science and Engineering

Submitted by

Akshar Vibhav(4NI16CS009)
Ashwini R (4NI16CS020)
Bhargavi R Kamat(4NI16CS023)
Bindu R N(4NI16CS025)

Under the guidance of

Ms. Padmini M S

Assistant Professor

Department of Computer Science and Engineering
The National Institute of Engineering
(Autonomous under VTU) Manandavadi Road, Mysuru-570008

2019-2020

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

THE NATIONAL INSTITUTE OF ENGINEERING

CERTIFICATE

This is to Certify that the project work entitled "**KUZUSHIJI REOGNITION"** is a bonafide work carried out by **Akshar Vibhav** (4NI16CS009), **Ashwini R** (4NI16CS020), **Bhargavi R Kamat** (4NI16CS023) and **Bindu R N** (4NI14CS025) in partial fulfillment for project work – I, eighth semester, Computer Science and Engineering**,** The National Institute of Engineering (Autonomous under VTU) during the academic year 2019 – 2020. It is certified that all corrections and suggestions indicated for the Internal Assessment have been incorporated in the report deposited in the department library. The project work – II report has been approved in partial fulfillment as per academic regulations of The National Institute of Engineering, Mysuru**.**

Internal Guide

……

(Ms. Padmini M.S)
Assistant Professor

Professor & Head

……..

(Dr. V. K. Annapurna)
Professor & Head,
Dept of CSE

Principal

…….....

(Dr. Rohini Nagapadma)
Principal

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without the mention of people whose ceaseless cooperation made it possible whose constant guidance and encouragement crown all efforts.

First and foremost we would like to thank our beloved principal **Dr. Rohini Nagapadma** for being the patron and the beacon light for this project.

We would like to express our sincere gratitude to our HOD **Dr. V. K. Annapurna,** HOD, Department of CSE, NIE for her relentless support and encouragement.

It gives us immense pleasure to thank our guide **Ms. Padmini M.S,** Assistant Professor, Department of CSE, NIE for her valuable suggestions and guidance during the process of the seminar and for having permitted us to pursue work on the subject.

ABSTRACT

Japan has millions of books and over a billion historical documents such as personal letters or diaries preserved nationwide. Most of them cannot be read by the majority of Japanese people living today because they were written in “Kuzushiji”. Due to the lack of available human resources, there has been a great deal of interest in using Machine Learning to automatically recognize these historical texts and transcribe them into modern Japanese characters. The model, as an add-on feature, optionally, would also try to convert the Japanese characters to English characters.

TABLE OF CONTENTS

Chapter 1

INTRODUCTION

Kuzushiji is a cursive script that was used in Japan for over a thousand years that has fallen out of use in modern times. Vast portions of Japanese historical documents now cannot be read by most Japanese people. In Kuzushiji documents, Kanji, Hiragana and Katakana are all used. However, the number of character types in each document varies by genre. For example, story books are mostly written in Hiragana while formal records are written mainly in Kanji. Many challenges were faced during Kuzushiji recognition. The first one being the large number of character types. The total number of unique characters in the Kuzushiji dataset is over 4300. However, the frequency distribution is very long-tailed and a large fraction of the characters may only appear once or twice in a book. Therefore, the dataset is highly unbalanced. Secondly, the quintessential characteristic of Classical Hiragana or Hentaigana is that many characters which can only be written a single character in modern Japanese can be written in many different ways in Kuzushiji. Kuzushiji was written in a cursive script, and hence in many cases, characters are connected or overlap which can make the recognition task difficult. Even though Kuzushiji, a cursive writing style, had been used in Japan for over thousands of years, there are very few fluent readers of Kuzushiji today (only 0.01% of modern Japanese natives). Due to the lack of available human resources, there has been a great deal of interest in using Machine Learning to automatically recognize these historical texts and transcribe them into modern Japanese characters. Nevertheless, several challenges in Kuzushiji recognition have made the performance of existing systems extremely poor.

Chapter 2

SYSTEM ANALYSIS

2.1 LITERATURE SURVEY

2.1.1 Research Papers on Existing System

[1] Anh Duc Le, Tarin Clanuwat, Asanobu Kitamoto;" **A human-Inspired Recognition System for Pre-Modern Japanese Historical Documents"**, IEEE 2019

Description:

A human-inspired recognition system was proposed in order to recognize Kuzushiji documents. This system simulates the human reading behaviour which is able to determine the start character of a text line, scan the next character and determine the end of a text line to move to the next text line.

[2] Chulapong Panichkriangkrai; "**Internet-Based Interactive Transcription Support System for Woodblock-Printed Japanese Historical Book Images**"; IIAI-AAI 2018

Description:

It is an Internet-based interactive transcription support system. It helps users to do character segmentation, modify character segmentation, and make transcription. In future, the usability of the system could be improved.

[3] Yuta Hashimoto, Yoichi Iikura; "**The Kuzushiji Project: Developing a Mobile Learning Application for Reading Early Modern Japanese Texts**"; DHQ 2017

Description**:**

A mobile-learning application was developed for reading as an aid for learning Kuzushiji characters and documents. The application has three modules: character reading, text reading, and community modules. The character module has flashcard-like features for users to learn character shapes of 102 hentaigana and 176 kanji characters. The reading module helps users to learn how to read Kuzushiji characters from real classical texts. The classical texts are accompanied by transcriptions. Users can look up the transcriptions

when they cannot read the text. The community module lets users communicate with each other to exchange learning materials and learning experiences.

 K. Ikeda, R. Hayashi, K. Nagasaki, A. Morishima; "**Human-assisted OCR of Japanese books with different kinds of microtasks**"; iSchools, iConference 2017

Description**:**

Human-assisted OCR is a common approach for transcribing books and has been used for many digital library projects. The system is built for transcribing the book collections of National Diet Library. The human-assisted OCR approach was extended to distribute microtasks in many ways other than just showing tasks in the specific Web page.

[5]Tadashi Horiuchi; **"A Study on Japanese Historical Character Recognition Using Modular Neural Networks**"; ICICIC 2009

Description**:**

Modular neural networks is used for recognizing Kuzushiji characters, it consists of a rough-classifier and a set of fine-classifiers to recognize Kuzushiji script. The system employed Multi-Templates Matching for the rough-classifier and Multi layered Perceptrons for the fine-classifier — first, the rough-classifier selects several candidates of character categories for an input pattern. Then, the fine-classifier finds the best result from the candidates.

2.1.2. EXISTING SYSTEM:

The existing human-inspired document-reading system recognizes multiple lines of pre-modern Japanese historical documents. They can also determine the end of a line or skip a figure to move to the next line. First, the recognition system detects where to start a text line. Second, the system scans and recognize character by character until the text line is completed. Then, the system continues to detect the start of the next text line. This process is repeated until reading the whole document. Though the system supports multiple lines, the performance of the model to handle long range context, large vocabularies, and non-standardized character layouts is poor.

2.1.3. PROPOSED SYSTEM:

The proposed system is of two stages: Detection and Classification.

- Detection:

  For detecting the characters in a picture,one of the well-known keypoint-based detectors is CornerNet is used. The concept of ConerNet is predicting the corner of the bounding boxes using heatmaps like semantic segmentation. This detector is relatively easy to implement since it does not need multiple anchors unlike other well-known one-stage detectors such as RCNN, YOLO and SSD.

  Later, CenterNet (Object as Points), which is one of the keypoint based detectors derieved from CornerNet is applied to overcome few of the restrictions caused due to cornernet. CenterNet directly predicts the center pixel of the object with heatmap.

- Classification:

  For classification, CNN model is used. A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other.

It contains various layers of which some of them are *convolutional layer, pooling layer, activation layer, fully connected layer.*

- The convolution layer is the main building block of a convolutional neural network. The input is a tensor with shape (number of images) x (image width) x (image height) x (image depth). Then after passing through a convolutional layer, the image becomes abstracted to a feature map, with shape (number of images) x (feature map width) x (feature map height) x (feature map channels).
  Convolutional layers convolve the input and pass its result to the next layer.

- Activation Layer is an activation function that decides the final value of a neuron. ReLu refers to the Rectifier Unit, the most commonly deployed activation function for the outputs of the CNN neurons. LeakyReLu is used here as an activation function.

- Pooling layers reduce the dimensions of the data by combining the outputs of neuron clusters at one layer into a single neuron in the next layer. Local pooling combines small clusters, typically 2 x 2. Global pooling acts on all the neurons of the convolutional layer. In addition, pooling may compute a max or an average. *Max pooling* uses the maximum value from each of a cluster of neurons at the prior layer. *Average pooling* uses the average value from each of a cluster of neurons at the prior layer. In the model, average pooling is used.

- In the fully connected layer, the neurons have a complete connection to all the activations from the previous layers. Their activations can hence be computed with a matrix multiplication followed by a bias offset. This is the last phase for a CNN network. The flattened matrix goes through a fully connected layer to classify the images.

This model overcomes the drawback of the existing model. That is, this model is capable of handling long range context, large vocabularies, and non-standardized character layouts.

2.2 SYSTEM REQUIREMENTS

Programming Language : Python

Drivers : NA

Operating System : Windows/Linux

Tools : Anaconda, matplotlib, keras, sklearn, pandas, tensorflow, numpy

Processor : Dual core, 64-bit processor

Memory : 4GB or higher

Chapter 3

SYSTEM DESIGN DETAILS

3.1 System Architecture

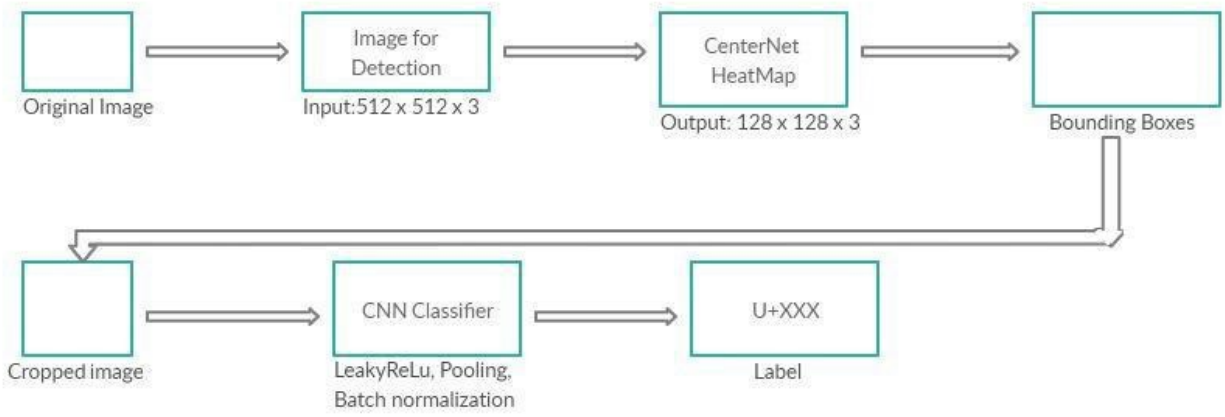

Fig 3.1: System architecture

3.2 Architecture Overview:

1. DataSet:

- train.csv - the training set labels and bounding boxes.
    - image_id: the id code for the image.
    - labels: a string of all labels for the image. The string should be read as space separated series of values where Unicode character, X, Y, Width, and Height are repeated as many times as necessary.

train.csv (14.95 MB)

	image_id	labels
	3605 unique values	3605 unique values
1	100241706_00004_2	U+306F 1231 3465 133 53 U+304C 275 1652 84 69 U+3044 1495 1218 143 69 U+3051 220 3331 53 91 U+306B 911 1452 61 92 U+306B 927 3445 71 92 U+306E 904 2879 95 92 U+5DE5 1168 1396 187 95 U+3053 289 3166 69 . . .
2	100241706_00005_1	U+306F 1087 2018 103 65 U+304B 1456 1832 40 73 U+304B 2036 1722 65 76 U+3044 1709 1567 136 76 U+306B 1741 3329 43

Fig 3.2(a) : train.csv

- unicode_translation.csv - supplemental file mapping between unicode IDs and Japanese characters.

unicode_translation.csv (51.41 KB)

	Unicode	char
	4781 unique values	4777 unique values
1	U+0031	1
2	U+0032	2
3	U+0034	4
4	U+0036	6
5	U+0039	9
6	U+003F	?
7	U+2000B	𠂉
8	U+20D45	𠵅
9	U+2123D	𡈽
10	U+22999	𢦙
11	U+22CE3	𢳣
12	U+231C3	𣇃

Fig 3.2(b) Unicode_translation.csv

- [train/test]_images.zip - the images.

  Each image consists of many characters in it. There can be non-characters too.

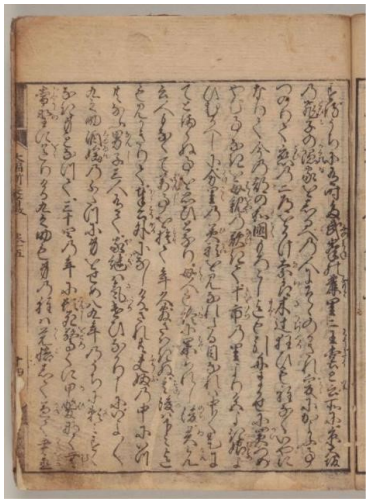

Fig 3.2(c) Test/Train image

2. Original Image:

An image consists of several characters of various sizes. So, its necessary to know the average size of the characters present in each image to make the processing easy and accurate.

3. Image for Detection:

Here, the image is made ready to be sent as input to the CenterNet. Image is resized to 512 x 512 x 3.

4. CenterNet:

- CornerNet uses hourglass network as the backbone, followed by two prediction modules. One module is for the top-left corners, while the other one is for the bottom-right corners. Each module consists of its own corner pooling module to pool features from the hourglass network before predicting the heatmaps, embeddings, and offsets. But still, the performance of CornerNet is restricted when detecting the boundaries of the objects. CenterNet tries to overcome this restriction. CenterNet detects the center heatmap. It is used to remap the corners from the heatmaps of the input image.
- A heatmap is a graphical representation where individual values of a matrix are represented as colors. A heatmap is very useful in visualizing the concentration of values between two dimensions of a matrix. This helps in finding patterns and gives a perspective of depth.

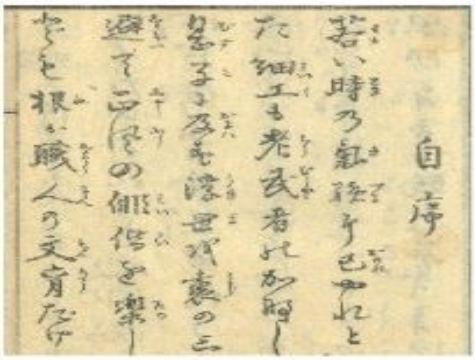

Fig 3.2(d)

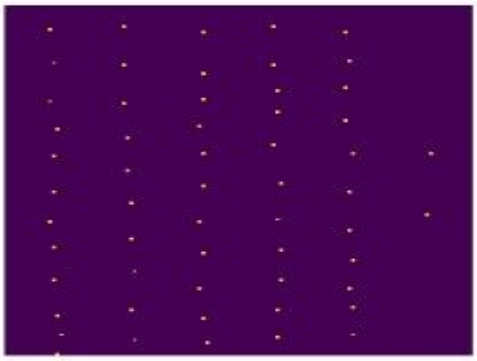

Fig 3.2(e)

- Figure 3.2(d) is the original image. Figure 3.2(e) is target image in which center points are 1, other pixcels are 0.

5. Bounding Boxes:

The output is converted to bounding boxes. NMS (Non Maximum Supression) is used to find the best box. NMS makes sure that a particular object is identified only once.

6.CNN Classifier:

CNN model is built and used here for classification.

- A Convolutional Neural Network (CNN) is a deep learning algorithm that can recognize and classify features in images for computer vision. It is a multi-layer neural network designed to analyze visual inputs and perform tasks such as image classification, segmentation and object detection, which can be useful for autonomous vehicles.
- There are two main parts to a CNN:
    - A convolution tool that splits the various features of the image for analysis
    - A fully connected layer that uses the output of the convolution layer to predict the best description for the image.

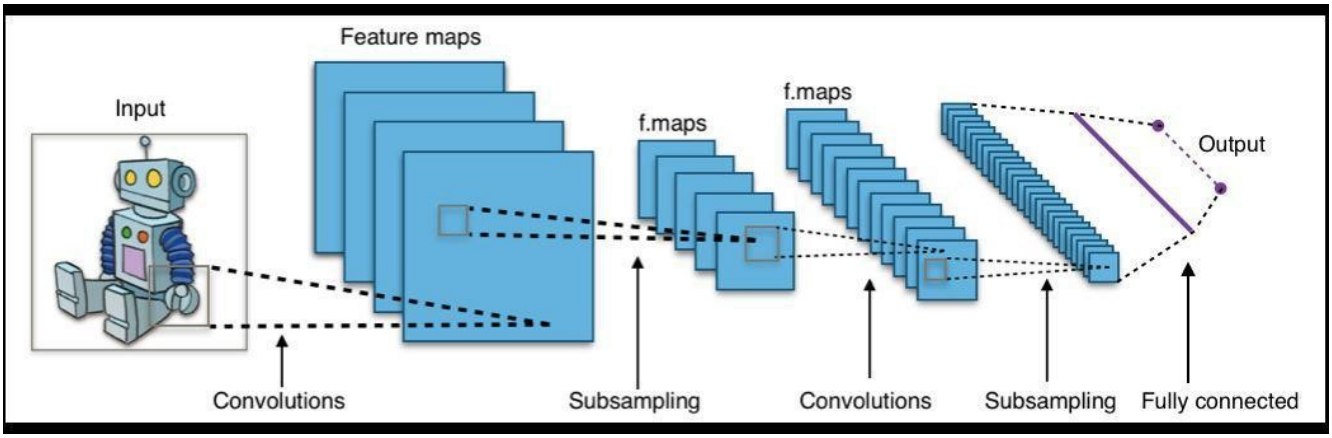

Fig 3.2(f) CNN architecture

- It contains various layers of which some of them are *convolutional layer, pooling layer, activation layer, fully connected layer.*
- The convolution layer is the main building block of a convolutional neural network. The input is a tensor with shape (number of images) x (image width) x (image height) x (image depth). Then after passing through a convolutional layer, the image becomes abstracted to a feature map, with shape (number of images) x (feature map width) x (feature map height) x (feature map channels). Convolutional layers convolve the input and pass its result to the next layer.
- Activation Layer is an activation function that decides the final value of a neuron. ReLu refers to the Rectifier Unit, the most commonly deployed activation function for the outputs of the CNN neurons. LeakyReLu is used here as an activation function.
- Pooling layers reduce the dimensions of the data by combining the outputs of neuron clusters at one layer into a single neuron in the next layer. Local pooling combines small clusters, typically 2 x 2. Global pooling acts on all the neurons of the convolutional layer. In addition, pooling may compute a max or an average. *Max pooling* uses the maximum value from each of a cluster of neurons at the prior layer. *Average pooling* uses the average value from each of a cluster of neurons at the prior layer. In the model, average pooling is used.

- In the fully connected layer, the neurons have a complete connection to all the activations from the previous layers. Their activations can hence be computed with a matrix multiplication followed by a bias offset. This is the last phase for a CNN network. The flattened matrix goes through a fully connected layer to classify the images.

CHAPTER 4

SYSTEM IMPLEMENTATION

4.1 Preprocessing (Check Object Size)

- The first step is to determine the size of the objects. Since the characters are of different sizes, it is necessary to know the average size of the objects. If the object size is too small, detector cannot find the target. The graph of the ratio of letter size to picture size is plotted.

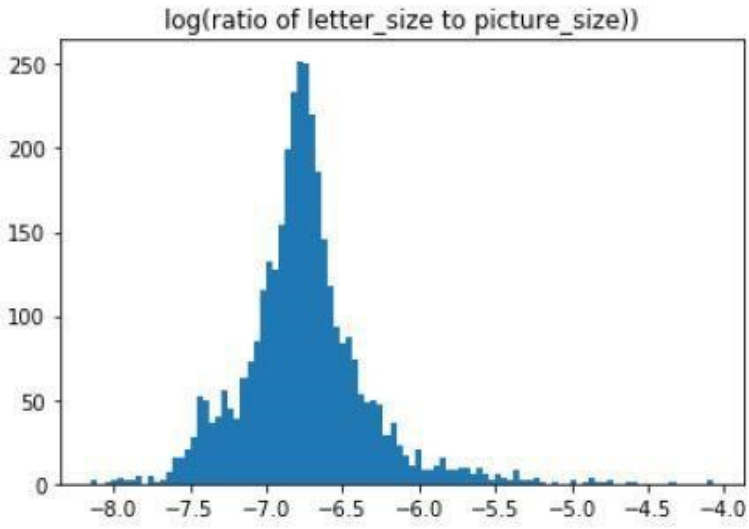

Fig 4.1(a) Graph of letter size to picture size

- It can be seen that the average size of the objects varies among pictures. So it would be better to split the picture into several parts. So cropping pictures with appropriate ratio would be important when detecting letters. One way to find the best cropping size is creating the model to predict the average letter size.
- CNN model is created, which is almost a ResNet model. The input to the model will be image resized to 512 x 512 x 3.
- CNN model has many layers. First being the convolution layer, the images are cropped randomly.

```
#random crop
if random_crop and is_train:
  pic_width,pic_height=f.size
  f=np.asarray(f.convert('RGB'),dtype=np.uint8)
  top_offset=np.random.randint(0,pic_height-int(crop_ratio*pic_height))
  left_offset=np.random.randint(0,pic_width-int(crop_ratio*pic_width))
  bottom_offset=top_offset+int(crop_ratio*pic_height)
  right_offset=left_offset+int(crop_ratio*pic_width)
  f=cv2.resize(f[top_offset:bottom_offset,left_offset:right_offset,:],(input_height,input_width))
else:
  f=f.resize((input_width, input_height))
  f=np.asarray(f.convert('RGB'),dtype=np.uint8)
```

Fig 4.1(b) Random cropping of images

- Next comes the activation layer. The activation function is a node that is put at the end of or in between Neural Networks. They help to decide if the neuron would fire or not. The activation function is the non linear transformation that we do over the input signal. This transformed output is then sent to the next layer of neurons as input.
- Here, LeakyReLu is used as an activation function. Leaky ReLUs are one attempt to fix the “dying ReLU” problem. Instead of the function being zero when x < 0, a leaky ReLU will instead have a small negative slope.
- BatchNormalization is also applied. Batch normalization is a technique for training very deep neural networks that standardizes the inputs to a layer for each mini-batch. This has the effect of stabilizing the learning process and dramatically reducing the number of training epochs required to train deep networks.

```
x_deep= Conv2DTranspose(deep_ch, kernel_size=2, strides=2, padding='same', use_bias=False)(x_deep)
x_deep = BatchNormalization()(x_deep)
x_deep = LeakyReLU(alpha=0.1)(x_deep)
x = Concatenate()([x_shallow, x_deep])
x=Conv2D(out_ch, kernel_size=1, strides=1, padding="same")(x)
x = BatchNormalization()(x)
x = LeakyReLU(alpha=0.1)(x)
return x
```

Fig 4.1(c) Activation and batch normalization

- Next pooling is done. A pooling layer is another building block of a CNN. Its function is to progressively reduce the spatial size of the representation to reduce the amount of

parameters and computation in the network. Pooling layer operates on each feature map independently. Here, average pooling is applied

```
input_layer_1=AveragePooling2D(2)(input_layer)
input_layer_2=AveragePooling2D(2)(input_layer_1)
```

Fig 4.1(d) Pooling layer

- Then the model created is fitted. Model fitting is a measure of how well a machine learning model generalizes to similar data to that on which it was trained. During the fitting process, an algorithm is run on data for which the target variable is known, that is "labeled" data, and produce a machine learning model.
- An epoch is a hyperparameter which is defined before training a model. One epoch is when an entire dataset is passed both forward and backward through the neural network only once. One epoch is too big to feed to the computer at once. So, it is divided to several smaller batches.
- Batch size refers to the number of training examples utilized in one iteration.
- The verbose flag is set to 1 here, which specifies whether the detailed information is to be printed in the console about the progress of the training.

```
hist = model.fit_generator(
    Datagen_sizecheck_model(train_list,batch_size, is_train=True,random_crop=True),
    steps_per_epoch = len(train_list) // batch_size,
    epochs = n_epoch,
    validation_data=Datagen_sizecheck_model(cv_list,batch_size, is_train=False,random_crop=False),
    validation_steps = len(cv_list) // batch_size,
    callbacks = [lr_schedule, model_checkpoint],#[early_stopping, reduce_lr, model_checkpoint],
    shuffle = True,
    verbose = 1
)
return hist
```

Fig 4.1(e) Fitting the Model

- Next, the model is allowed to learn. Learning rate is specified. Weights of every node in the various layers of the model is determined and is stored in a file, final_weights_step1.hdf5.

- Learning rate is a parameter that controls how much the weights of the network is adjusted with respect to loss gradient.
- Weights determine the strength of the connection of the neurons.It connects each neuron in one layer to every neuron in the next layer.
- A traditional default value for learning rate is 0.1 or 0.01.
- Larger learning rates will require fewer training epochs.
- If learning rate is set too low, training will be too slow as it will be making very tiny updates to the weights. If its high, it can cause undesirable divergent behavior in loss function.
- Here, the default learning rate is made 0.0005 and if the epoch is greater than 10, then the learning rate is made 0.0001.

```
K.clear_session()
model=create_model(input_shape=(input_height,input_width,3),size_detection_mode=True)
"""
# EarlyStopping
early_stopping = EarlyStopping(monitor = 'val_loss', min_delta=0, patience = 10, verbose = 1)
# ModelCheckpoint
weights_dir = '/model_1/'
if os.path.exists(weights_dir) == False:os.mkdir(weights_dir)
model_checkpoint = ModelCheckpoint(weights_dir + "val_loss{val_loss:.3f}.hdf5", monitor = 'val_loss', verbose = 1,
                                   save_best_only = True, save_weights_only = True, period = 1)
# reduce learning rate
reduce_lr = ReduceLROnPlateau(monitor = 'val_loss', factor = 0.5, patience = 10, verbose = 1)
"""
def lrs(epoch):
    lr = 0.0005
    if epoch>10:
        lr = 0.0001
    return lr

lr_schedule = LearningRateScheduler(lrs)
model_checkpoint = ModelCheckpoint("final_weights_step1.hdf5", monitor = 'val_loss', verbose = 1,
                                   save_best_only = True, save_weights_only = True, period = 1)
print(model.summary())
```

Fig 4.1(f) Learning Rate

- Model summary is printed where the first column is the layer, second is the output shape, third is the number of parameters and the forth is the connected to column specifying the layer to which it is connected.

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 512, 512, 3)	0	
conv2d_1 (Conv2D)	(None, 256, 256, 16)	448	input_1[0][0]
batch_normalization_1 (BatchNor	(None, 256, 256, 16)	64	conv2d_1[0][0]
leaky_re_lu_1 (LeakyReLU)	(None, 256, 256, 16)	0	batch_normalization_1[0][0]
average_pooling2d_1 (AveragePoo	(None, 256, 256, 3)	0	input_1[0][0]
concatenate_1 (Concatenate)	(None, 256, 256, 19)	0	leaky_re_lu_1[0][0] average_pooling2d_1[0][0]
conv2d_2 (Conv2D)	(None, 128, 128, 32)	5504	concatenate_1[0][0]

Fig 4.1(g) Model Summary

- Next, predicting the created model is done. Prediction is useful because it gives accurate insight into any question and allows users to forecasts.

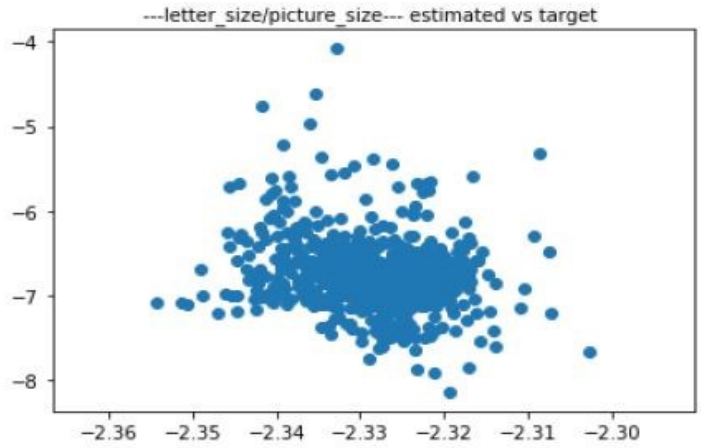

Fig 4.1(h)

From the above graph, it can be seen that the training is not that great. Anyway, this can be used further because of the hardware limitations for training and testing the models.

4.2 Detection By CenterNet

- For detection, concept of CenterNet and heatmaps are used.
- CenterNet detects the center heatmap. It is used to remap the corners from the heatmaps of the input image.
- A heatmap is a graphical representation where individual values of a matrix are represented as colors. A heatmap is very useful in visualizing the concentration of values between two dimensions of a matrix. This helps in finding patterns and gives a perspective of depth.
- First the model for detection has to be made. The input data is made ready by resizing the images to 512 x 512 x 3. The output of the model will be Heatmap of center point 128x128x1.

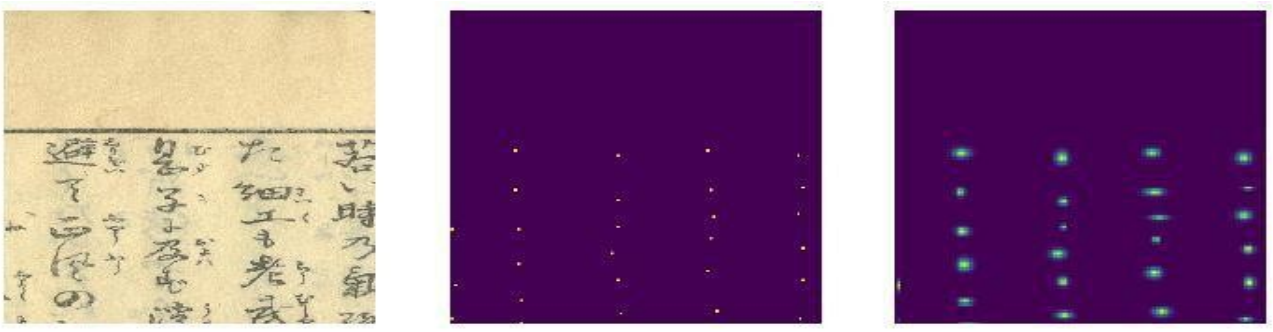

Fig 4.2(a) Heatmaps

- The figure 4.2(a) gives an example of how the letters are detected using CenterNet.
- Left image is the original picture(cropped).
- Middle one is the target image in which center points are 1, other pixcels are 0.
- Right one shows center points with gaussian distributions.
- Right image is necessary to reduce the training loss when the model detect the points near the exact center.
- Based on CenterNet and heatmap, a model is created and fitted.

```
    category=0#not classify, just detect
    heatmap=((np.exp(-(((np.arange(output_width)-x_c)/(width/10))**2)/2)).reshape(1,-1)
                        *(np.exp(-(((np.arange(output_height)-y_c)/(height/10))**2)/2)).reshape(-1,1))
    output_layer[:,:,category]=np.maximum(output_layer[:,:,category],heatmap[:,:])
    output_layer[int(y_c//1),int(x_c//1),category_n+category]=1
    output_layer[int(y_c//1),int(x_c//1),2*category_n]=y_c%1#height offset
    output_layer[int(y_c//1),int(x_c//1),2*category_n+1]=x_c%1
    output_layer[int(y_c//1),int(x_c//1),2*category_n+2]=height/output_height
    output_layer[int(y_c//1),int(x_c//1),2*category_n+3]=width/output_width
  y.append(output_layer)

    hist = model.fit_generator(
        Datagen_centernet(train_list,batch_size),
        steps_per_epoch = len(train_list) // batch_size,
        epochs = n_epoch,
        validation_data=Datagen_centernet(cv_list,batch_size),
        validation_steps = len(cv_list) // batch_size,
        callbacks = [lr_schedule],#early_stopping, reduce_lr, model_checkpoint],
        shuffle = True,
        verbose = 1
```

Fig 4.2(b)

- Figure 4.2(b) shows the code snippets of heatmap function and fitting the created model respectively.
- Next, the model is made to learn by specifying the learning rate and the model summary is printed for better understanding.

```
K.clear_session()
model=create_model(input_shape=(input_height,input_width,3),size_detection_mode=False)

def lrs(epoch):
    lr = 0.001
    if epoch >= 20: lr = 0.0002
    return lr

lr_schedule = LearningRateScheduler(lrs)

"""

# EarlyStopping
early_stopping = EarlyStopping(monitor = 'val_loss', min_delta=0, patience = 60, verbose = 1)
# ModelCheckpoint
weights_dir = '/model_2/'

if os.path.exists(weights_dir) == False:os.mkdir(weights_dir)
model_checkpoint = ModelCheckpoint(weights_dir + "val_loss{val_loss:.3f}.hdf5", monitor = 'val_loss', verbose = 1,
                                   save_best_only = True, save_weights_only = True, period = 3)
# reduce learning rate
reduce_lr = ReduceLROnPlateau(monitor = 'val_loss', factor = 0.5, patience = 10, verbose = 1)
"""
model.load_weights('final_weights_step1.h5',by_name=True, skip_mismatch=True)

print(model.summary())
```

Fig 4.2(d)

- The figure 4.2(d) shows that the model is created, made to learn and the weights are stored in file, final_weights_step1.h5.

batch_normalization_51 (BatchNo	(None, 128, 128, 5)	20	conv2d_45[0][0]
up_sampling2d_4 (UpSampling2D)	(None, 128, 128, 5)	0	leaky_re_lu_56[0][0]
leaky_re_lu_51 (LeakyReLU)	(None, 128, 128, 5)	0	batch_normalization_51[0][0]
concatenate_12 (Concatenate)	(None, 128, 128, 10)	0	up_sampling2d_4[0][0] leaky_re_lu_51[0][0]
conv2d_51 (Conv2D)	(None, 128, 128, 5)	455	concatenate_12[0][0]
activation_1 (Activation)	(None, 128, 128, 5)	0	conv2d_51[0][0]

Total params: 25,853,915
Trainable params: 25,836,467
Non-trainable params: 17,448

Fig 4.2(e)

- The figure 4.2(e) shows the model summary where first column is the layer, second column is the output size, third is the number of parameters and forth column specifies the layer to which it is connected.
- It also gives the count of total parameters, trainable parameters and non-trainable parameters.
- After the model is created, it is necessary to train the model.
- The data is split as test data and train data. The test split represent the proportion of dataset to be included for testing purpose.
- Here, in this model, 20% of the data is taken for test purpose and the rest for training purpose.
- Optimizer is used. Optimizer shape and mold the model into its most accurate possible form using the weights.
- Here, optimizer called Adam is used since it can handle sparse gradients on noisy problems.
- The optimizer is used by passing it to model.complie.

```
train_list, cv_list = train_test_split(annotation_list_train_w_split, random_state = 111,test_size = 0.2)#stratified split is bett
er

learning_rate=0.01
n_epoch=1
batch_size=32
model.compile(loss=all_loss, optimizer=Adam(lr=learning_rate), metrics=[heatmap_loss,size_loss,offset_loss])
hist = model_fit_centernet(model,train_list,cv_list,n_epoch,batch_size)

model.save_weights('final_weights_step2.h5')
```

```
Epoch 1/1
90/90 [==============================] - 2629s 29s/step - loss: 4.0059 - heatmap_loss: 2.2906 - size_loss: 1.2082 - o
ffset_loss: 0.5071 - val_loss: 8.8221 - val_heatmap_loss: 4.4554 - val_size_loss: 3.7206 - val_offset_loss: 0.5084
```

Fig 4.2(f)

- This model runs the whole data set once. It takes 2629 seconds to do this and the various other losses are printed too.
- The next step involves converting the output obtained to bounding boxes. For this, NMS is used.
- NMS (Non Maximum Supression) is used to find the best box. NMS makes sure that a particular object is identified only once.

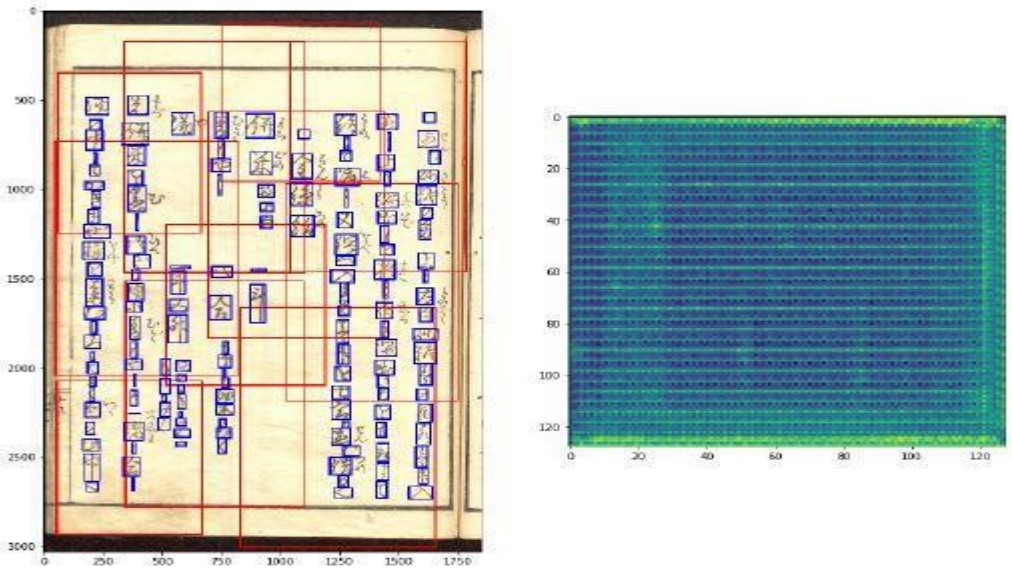

Fig 4.2(g)

- The blue boxes are drawn wherever a character is present in the image.

4.3 Classification

- After detecting the characters in an image using centernet and heatmaps, and also after creating bounding boxes using NMS, now these characters need to be identified and classified.
- The images are cropped in advanced and they are sent to the CNN model created earlier.
- The classification model is built and fitted.
- Later it is trained with the trained input, which gives the accuracy of 15%.
- The accuracy could have been more high if there were GPUs to train the model.
    - After the training, the weights are stored in the file, final_weights_step3.h5.

```
train_list, cv_list = train_test_split(train_input, random_state = 111,test_size = 0.2)
learning_rate=0.5
n_epoch=1
batch_size=64

model.compile(loss="categorical_crossentropy", optimizer=Adam(lr=learning_rate),metrics=["accuracy"])
hist = model_fit_classification(model,train_list,cv_list,n_epoch,batch_size)

model.save_weights('final_weights_step3.h5')
```

```
Epoch 1/1
8543/8543 [==============================] - 25662s 3s/step - loss: 52.1513 - accuracy: 0.1545 - val_loss: 208.6963 -
val_accuracy: 0.0147
```

Fig 4.3(a)

- Then the result is checked for few characters, where the character is detected and its corresponding Japanese character is printed.

```
for i in range(3):
  img = np.asarray(Image.open(train_input[i][0]).resize((32,32)).convert('RGB'))
  predict=np.argmax(model.predict(img.reshape(1,32,32,3)/255),axis=1)[0]
  name = dict_translation[inv_dict_cat[str(predict)]]
  print(name)
  plt.imshow(img)
  plt.show()
```

Fig 4.3(b)

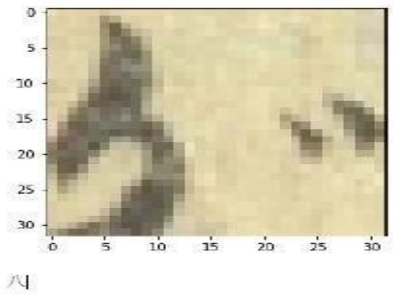

Fig 4.3(c)

- The figure 4.3(c) shows an example where the corresponding Japanese character of the image is recognized.
- Later, all the weights are made use. The weights calculated in the model 1, where activation function, cropping and pooling was done, the weights calculated in the model2 where detection was made using CenterNet and heatmaps, and the weights calculated in the model3 where classification was done are used. The weights of these three models are pipelined.
- A pipeline is used to help automate machine learning workflows. They operate by enabling a sequence of data to be transformed and correlated together in a model that can be tested and evaluated to achieve an outcome, whether positive or negative.

```
model_1=create_model(input_shape=(512,512,3),size_detection_mode=True)
model_1.load_weights('final_weights_step1.h5')
#model_1.load_weights('final_weights_step1.hdf5')

model_2=create_model(input_shape=(512,512,3),size_detection_mode=False)
model_2.load_weights('final_weights_step2.h5')

model_3=create_classification_model(input_shape=(32,32,3),n_category=len(dict_cat))
model_3.load_weights('final_weights_step3.h5')


def pipeline(i,print_img=False):
  # model1: determine how to split image
  if print_img: print("model 1")
  img = np.asarray(Image.open(id_test[i]).resize((512,512)).convert('RGB'))
  predicted_size=model_1.predict(img.reshape(1,512,512,3)/255)
  detect_num_h=aspect_ratio_pic_all_test[i]*np.exp(-predicted_size/2)
  detect_num_w=detect_num_h/aspect_ratio_pic_all_test[i]
  h_split_recommend=np.maximum(1,detect_num_h/base_detect_num_h)
  w_split_recommend=np.maximum(1,detect_num_w/base_detect_num_w)
  if print_img: print("recommended split_h:{}, split_w:{}".format(h_split_recommend,w_split_recommend))

  # model2: detection
  if print_img: print("model 2")
  img=Image.open(id_test[i]).convert("RGB")
  box_and_score_all=split_and_detect(model_2,img,h_split_recommend,w_split_recommend,score_thresh=0.3,iou_thresh=0.4)#output:scor
e,top,left,bottom,right
  if print_img: print("find {} boxes".format(len(box_and_score_all)))
  print_w, print_h = img.size
  if (len(box_and_score_all)>0) and print_img:
    img=draw_rectangle(box_and_score_all[:,1:],img,"red")
    plt.imshow(img)
    plt.show()

  # model3: classification
```

Fig 4.3(d)

- The final result with the test_image_id and all the Unicode values of the characters present in the image is stored in a csv file, Submission.csv.

```
for i in tqdm(range(len(id_test))):
  ans=pipeline(i,print_img=False)
  df_submission.at[i, 'labels']=ans

df_submission.to_csv("submission.csv",index=False)
```

Fig 4.3(e)

- Figure 4.3(f) shows the final result of few test_image_id and the corresponding Unicode values predicted.

```
In [65]:
df_submission.head()
Out[65]:
```

	image_id	labels	Useage
0	test_001c37e2	U+6E1B 1303 1062 U+6E1B 1176 2533 U+6E1B 1175 ...	Private
1	test_003aa33a	U+6E1B 1143 1791 U+6E1B 646 2165 U+6E1B 1454 1...	Public
2	test_00665e33	U+5927 1350 573 U+516B 759 708 U+7236 1065 124...	Private
3	test_0082d2cd	U+6708 2161 763 U+6E1B 1275 2004 U+7236 1275 7...	Private
4	test_0086f578	U+6E1B 478 2052 U+6E1B 1305 986 U+6E1B 707 276...	Private

Fig 4.3(f)

- All the test_image with the Unicode values are stored in submission.csv file.

CHAPTER 5

CONCLUSION

The project completed its objective of recognizing and converting the Kuzushiji characters to the modern Japanese characters. The scope for future enhancements include improving the accuracy of the model and also adding a feature that converts the modern Japanese letters into English alphabets.

REFERENCES

[1] Anh Duc Le, Tarin Clanuwat, Asanobu Kitamoto;" **A human-Inspired Recognition System for Pre-Modern Japanese Historical Documents"**, IEEE 2019

[2] Chulapong Panichkriangkrai; "**Internet-Based Interactive Transcription Support System for Woodblock-Printed Japanese Historical Book Images**"; IIAI-AAI 2018

[3] Yuta Hashimoto, Yoichi Iikura; "**The Kuzushiji Project: Developing a Mobile Learning Application for Reading Early Modern Japanese Texts**"; DHQ 2017

[4] K. Ikeda, R. Hayashi, K. Nagasaki, A. Morishima; "**Human-assisted OCR of Japanese books with different kinds of microtasks**"; iSchools, iConference 2017

[5] Tadashi Horiuchi; **"A Study on Japanese Historical Character Recognition Using Modular Neural Networks**"; ICICIC 2009

[6] https://www.kaggle.com/c/kuzushiji-recognition/overview