

Acad. Year
2021-22

**** Urdu Character Recognition ****

Urdu Character Recognition



Sabahat Zafar

Abdul Basit

Bilal Ali

Faculty of Information Technology

Department of Computer Science

Salim Habib University, Karachi, Pakistan

Year 2021/22

Urdu Character Recognition

Submitted by

Sabahat Zafar (F19CSC18)

Abdul Basit (F19CSC06)

Bilal Ali (F19CSC38)

Faculty of Information Technology

Department of Computer Science

Machine Learning Course Project Report Presented to

Dr. Rizwan Ahmed Khan

Faculty of Information Technology

Department of Computer Science

Salim Habib University, Karachi, Pakistan

Year 2022

Contents

1 Introduction

- 1.1 Problem Statement
- 1.2 Objective
- 1.3 Urdu Language

2 Literature Review

- 2.1 Recognition of Urdu Handwritten Characters Using Convolutional Neural Network (2019)
- 2.2 Urdu Optical Character Recognition Systems (2016)
- 2.3 Urdu handwritten text recognition (2020)

3 Methodology

4 Libraries

- 4.1 TensorFlow
 - 4.1.1 Why TensorFlow?
- 4.2 Keras
 - 4.2.1 Why do we need Keras
- 4.3 OpenCV
 - 4.3.1 What is Computer Vision
 - 4.3.2 What is OpenCV
- 4.4 OS

5 Importing Libraries and Dataset

- 5.1 Importing Libraries
- 5.2 Loading Dataset
- 5.3 Splitting dataset into training and testing
- 5.4 Viewing data

6 Normalizing

6.1 Resizing

7 Creating Neural Network

7.1 viewing summary

8 Training Model

8.1 Compiling and training the model

8.2 Adding drop out layer

8.3 Reducing convolution layers

8.4 Early stopping

9 Evaluating model

9.1 Predicting on test data

9.2 predicting on imported images

10 Graphical User Interface (GUI)

11 References

1 Introduction

Handwritten recognition is an engaged spot of research in pattern recognition and holds multiple application in industrial and professional applications. Many of these application contains forms processing in multiple public departments such as government, administrative, health and academic etc. Urdu character recognition interests computerised conversion of a source language into its symbolic representation. The language can be illustrated in its spatial or temporal form. The study of the handwritten text provide advancement to a number of beneficial application such as author profiling, named entity recognition, and recognition of overlapped characters etc

1.1 Problem Statement

Text recognition is a tool that is widely used for different languages nowadays. Currently, we face lots of issues with the recognition of the Urdu language as there is currently not as much work done on this particular domain. Reading Urdu is difficult as the alphabetical arrangement changes according to the different words.

1.2 Objective

With our project we aim to provide readability of separate characters in the Urdu language. Eventually down the line to take it further, we will also try to work on recognition of words that those characters make up.

1.3 Urdu Language

Urdu letters are written from right to left .The numbers are written from left to right. Which is why Urdu is one of the bidirectional languages. Urdu Language consists of thirty eight basic letters and ten numeric. This alphabet set is considered a superset of all Urdu script-based languages' alphabets, i.e. the Arabic script contains 28 while the Persian script composes 32 alphabets. Furthermore, the Urdu Language also contains alphabets to express the Hindi

phonemes. All the Urdu-script-based languages, such as Arabic, Persian have some unique characteristics, i.e.

1. the script of these languages is written from right to left in cursive style
2. the script of these languages is context-sensitive, i.e. written in the form of ligatures, a combination of a single or many alphabets.

2 Literature Review

We Researched on different articles in order to make our project better. Some of the articles are:

2.1 Recognition of Urdu Handwritten Characters Using Convolutional Neural Network (2019)

In this paper, they propose the use of the convolutional neural network to recognize the multi-font offline Urdu handwritten characters in an unconstrained environment. They also propose a novel dataset of Urdu handwritten characters since there is no publicly-available dataset of this kind. A series of experiments are performed on our proposed dataset. The accuracy achieved for character recognition is among the best while comparing with the ones reported in the literature for the same task.

2.2 Urdu Optical Character Recognition Systems (2016)

This paper gives an across-the-board comprehensive review and survey of the most prominent studies in the field of Urdu optical character recognition (OCR). This paper introduces the OCR technology and presents a historical review of the OCR systems, providing comparisons between the English, Arabic, and Urdu systems. Detailed background and literature have also been provided for Urdu script, discussing the script's past, OCR categories, and phases. This paper further reports all state-of-the-art studies for different phases, namely, image acqui-

sition, pre-processing, segmentation, feature extraction, classification/recognition, and post-processing for an Urdu OCR system.

2.3 Urdu handwritten text recognition (2020)

Work on the problem of handwritten text recognition in Urdu script has been an active research area. A significant progress is made in this interesting and challenging field in the last few years. In this study, the authors presented a comprehensive survey for a number of offline and online handwritten text recognition systems for Urdu script written in Nastaliq font style from 2004 to 2019.

3 Methodology

We performed multiple steps so that we reach to completion of our project. We have used a large data set of images for our training and testing. We have used Neural Network for feature extraction

4 Libraries

We have used Multiple Libraries in our code for better performance and as a requirement of our classifier

4.1 TensorFlow

TensorFlow is a open source library for fast numerical computing. It is a Python library for fast numerical computing created and released by Google. It is a foundation library that can be used to create Deep Learning models directly or by using wrapper libraries that simplify the process built on top of TensorFlow. TensorFlow bundles together a slew of machine learning and deep learning models and algorithms (aka neural networks) and makes them useful by way of common programmatic metaphors. It can train and run deep neural networks for handwritten

digit classification, image recognition, word embedding, recurrent neural networks, sequence-to-sequence models for machine translation.

4.1.1 Why TensorFlow?

The single biggest benefit TensorFlow provides for machine learning development is abstraction. The TensorBoard visualization suite lets you inspect and profile the way graphs run by way of an interactive, web-based dashboard.

4.2 Keras

Keras is an API designed for human beings, not machines. Keras follows best practices for reducing cognitive load: it offers consistent simple APIs, it minimizes the number of user actions required for common use cases, and it provides clear actionable error messages. It also has extensive documentation and developer guides. It is part of the TensorFlow library and allows you to define and train neural network models in just a few lines of code.

4.2.1 Why do we need Keras

1. It offers consistent simple APIs, reduces the actions required to implement common code, and explains user error clearly.
2. It provides a variety of deployment options depending on user needs.
3. Keras is also deeply integrated with TensorFlow, so you can create customized workflows with ease.

4.3 OpenCV

4.3.1 What is Computer Vision

The term Computer Vision (CV) is used and heard very often in artificial intelligence (AI) and deep learning (DL) applications. The term essentially means giving a computer the ability to see the world as we humans do.

It is a field of study which enables computers to replicate the human visual system.

It's a subset of artificial intelligence which collects information from digital images or videos and processes them to define the attributes.

4.3.2 What is OpenCV

1. An open source software library for computer vision and machine learning.
2. OpenCV was created to provide a shared infrastructure for applications for computer vision and to speed up the use of machine perception in consumer products.
3. The library has more than 2500 optimised algorithms, including an extensive collection of computer vision and machine learning algorithms, both classic and state-of-the-art
4. Do complex tasks such as identify and recognise faces, identify objects, classify human actions in videos, track camera movements, track moving objects.

4.4 OS

Python OS module provides easy functions that allow us to interact and get Operating System information and even control processes up to a limit.

The OS module in Python provides functions for creating and removing a directory (folder), fetching its contents, changing and identifying the current directory, etc.

5 Importing Libraries and Dataset

5.1 Importing Libraries

```
: import tensorflow as tf
import numpy as np
import os
```

Figure 1: Libraries

5.2 Loading Dataset

```
: print(os.listdir("D:/sem6/ML/project"))
urdu_dataset = np.load('D:/sem6/ML/project/uhat_dataset.npz')
['archive.zip', 'data', 'Digits.ipynb.txt', 'New folder', 'Project Proposal.docx', 'uhat_dataset.npz']
```

Figure 2: loading dataset

5.3 Splitting dataset into training and testing

```
x_train = urdu_dataset['x_chars_train']
y_train = urdu_dataset['y_chars_train']

print('Size of train data for characters: ',x_train.shape)
print('Size of train labels for characters: ',y_train.shape)
Size of train data for characters: (28328, 28, 28)
Size of train labels for characters: (28328,)

x_test = urdu_dataset['x_chars_test']
y_test = urdu_dataset['y_chars_test']

print('Size of test data for characters: ',x_test.shape)
print('Size of test labels for characters: ',y_test.shape)
Size of test data for characters: (4880, 28, 28)
Size of test labels for characters: (4880,)
```

Figure 3: training and testing data (x) and classes (y)

5.4 Viewing data


```
import matplotlib.pyplot as plt
plt.gray()
plt.matshow(x_train[0])
plt.show()
```

<Figure size 432x288 with 0 Axes>

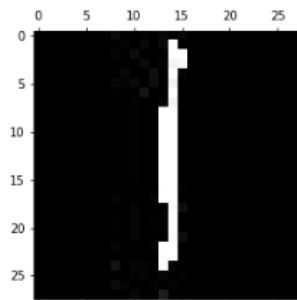


Figure 6: plotting x_train[0]
Plotting x_train[0] to view the image

```
: y_train[0]
: 0
```

Figure 7: y_train[0]
Getting label of x_train[0] through y_train[0] which gives output 0 corresponding to character 'Alif'

```
: romanNames=['alif','alif mad aa','bey','pey','tey','ttey',
              'sey','jeem','chay','bari he','khe','daal','dhaal','zaal',
              're','rhey','zaa','zhaa','seen','sheen','svaad','zvaad',
              'toy','zoy','ain','ghain','fey','qaaf','kaaf','gaaf','laam',
              'meem','noon','noonghunna','wao','choti he','do chashmi he',
              'ye','bari ye']
len(romanNames)
: 39
```

Figure 8: roman names array
The names of all the labels, which are the Urdu alphabets are stored in a romanNames array which is used to generate the roman names of the labels of testing and training data

6 Normalizing

As we saw, our data values are between 0-255 hence the need to normalize it and so all values fall between 0-1.

This is done through normalize function of Keras library. The function is essentially just performing $x_train/255$.

Normalizing our images is important so that if any of the colors change, we won't have to worry about it.

```
: x_train = tf.keras.utils.normalize(x_train, axis=1)
#same as x_train/255
```

Figure 9: normalizing

```
: plt.gray()
plt.matshow(x_train[0])
plt.show()

<Figure size 432x288 with 0 Axes>
```

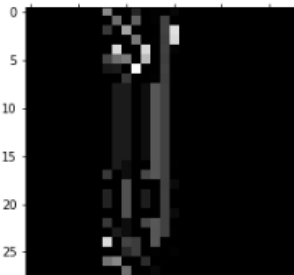


Figure 10: image after normalizing

```
#after normalization, all values b/w 0-1
x_train[0]
array([[0.      , 0.      , 0.      , 0.      , 0.      ,
        0.      , 0.      , 0.      , 0.4417261, 0.      ,
        0.      , 0.11642258, 0.      , 0.00870591, 0.      ,
        0.02245022, 0.      , 0.      , 0.      , 0.      ,
        0.      , 0.      , 0.      , 0.      , 0.      ,
        0.      , 0.      , 0.      ],
       [0.      , 0.      , 0.      , 0.      , 0.      ,
        0.      , 0.      , 0.      , 0.      , 0.34815531,
        0.      , 0.31046021, 0.      , 0.00979415, 0.21024167,
        0.      , 0.      , 0.      , 0.      , 0.      ,
        0.      , 0.      , 0.      ],
       [0.      , 0.      , 0.      , 0.      , 0.      ,
        0.      , 0.      , 0.      , 0.18405254, 0.      ,
        0.52674154, 0.      , 0.      , 0.      , 0.21024167,
        0.71560079, 0.      , 0.      , 0.      , 0.      ,
        0.      , 0.      , 0.      ],
       [0.      , 0.      , 0.      , 0.      , 0.      ,
        0.      , 0.      , 0.      , 0.      , 0.      ,
        0.      , 0.      , 0.      , 0.      , 0.      ,
        0.      , 0.      , 0.      ]])
```

Figure 11: x_train after normalizing

As we can see, now all our values are between 0 to 1.

6.1 Resizing

In order to apply convolution operation, the images need to be resized so that they have one more dimension.

We will resize all our training and testing images to add one more dimension so that now our images will have four dimensions instead of 3.

```
: img_size=28
x_train_resized=np.array(x_train).reshape(-1,img_size,img_size,1) #inc one dimension for kernel operation
x_test_resized=np.array(x_test).reshape(-1,img_size,img_size,1)

print("training samples dimensions: ", x_train_resized.shape)
print("testing samples dimensions: ", x_test_resized.shape)

training samples dimensions: (28328, 28, 28, 1)
testing samples dimensions: (4880, 28, 28, 1)
```

Figure 12: x_train and x_test after resizing

7 Creating Neural Network

We will import the sequential model. This will sequentially connect all our layers different layers which we have also imported Now, creating the network, each layer will first extract the

```
from tensorflow.keras import Sequential # sequentially connecting deep learning layers

from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten, Conv2D, MaxPooling2D
```

Figure 13: importing model and layers

features of the images through Conv2D.

Then the activation layer will drop the values that are less than 0 and only the values greater than 0 will move on to the next convolution layer.

MaxPooling2D will only take the maximum value 2x2 matrix to the preceding layers. Conv2D(64,(3,3), input_shape = (28,28,1))) means that 64 different kernels or filters will be used. Each will have a different matrix of size 3x3. So we'll be using 64 different 3x3 filters

```

: #creating neural network

model = Sequential()

#first convolution layer 0 1 2 3 (28,28,1) 28-3+1 = 26x26
model.add(Conv2D(64,(3,3), input_shape = (28,28,1))) #only for first conv layer to mention input layer size
model.add(Activation('relu')) #to make it non linear, drop values <0, next layer only values >0
model.add(MaxPooling2D(pool_size=(2,2))) #single maximum value of 2x2 matrix will get, rest will drop

#extracted feature then only allow +ve values then reduce the size by taking only maximum values to the next layer

#second conv layer 26-3+1=24x24 but maxpooling will reduce size so it'll be lower
model.add(Conv2D(64,(3,3), input_shape = (28,28,1)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

#third conv layer 24-3+1= 22x22
model.add(Conv2D(64,(3,3), input_shape = (28,28,1)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

#fully connected layer 20x20 = 400 each 400 will be connected to each of the 64
model.add(Flatten()) #before using this need to flatten so 2D is 1D
model.add(Dense(64)) #neural network layer, dense layer all neurons are connected
model.add(Activation('relu'))

#second fully connected layer. we're going down/reducing size to the number of classes
#model.add(Dense(32))
#model.add(Activation('relu'))

#last layer. output must be equal to number of classes of which we have 40
model.add(Dense(40))
model.add(Activation('softmax')) #sigmoid gives class probabilities for multiple classes

```

Figure 14: creating neural network
1 convolution layer having 64 different filters

7.1 viewing summary

```
: model.summary()
Model: "sequential_6"

```

Layer (type)	Output Shape	Param #
conv2d_11 (Conv2D)	(None, 26, 26, 64)	640
activation_18 (Activation)	(None, 26, 26, 64)	0
max_pooling2d_10 (MaxPooling2D)	(None, 13, 13, 64)	0
conv2d_12 (Conv2D)	(None, 11, 11, 64)	36928
activation_19 (Activation)	(None, 11, 11, 64)	0
max_pooling2d_11 (MaxPooling2D)	(None, 5, 5, 64)	0
conv2d_13 (Conv2D)	(None, 3, 3, 64)	36928
activation_20 (Activation)	(None, 3, 3, 64)	0
max_pooling2d_12 (MaxPooling2D)	(None, 1, 1, 64)	0
flatten_4 (Flatten)	(None, 64)	0
dense_8 (Dense)	(None, 64)	4160
activation_21 (Activation)	(None, 64)	0
dense_9 (Dense)	(None, 40)	2600
activation_22 (Activation)	(None, 40)	0

```

Total params: 81,256
Trainable params: 81,256
Non-trainable params: 0

```

Figure 15: model summary

We can see all the different layers in action. Notice how MaxPooling has reduced the size.

8 Training Model

8.1 Compiling and training the model

```
: print("total training samples: ", len(x_train_resized))  
total training samples: 28328
```

Figure 16: number of training samples

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

Figure 17: compiling the model

```
: history=model.fit(x_train_resized, y_train, epochs=5, validation_split=0.3) #training model  
Epoch 1/5  
620/620 [=====] - 22s 34ms/step - loss: 1.7364 - accuracy: 0.4664 - val_loss: 18.0665 - val_accuracy: 0.0027  
Epoch 2/5  
620/620 [=====] - 20s 33ms/step - loss: 0.8881 - accuracy: 0.7055 - val_loss: 17.6196 - val_accuracy: 0.0129  
Epoch 3/5  
620/620 [=====] - 21s 34ms/step - loss: 0.6644 - accuracy: 0.7769 - val_loss: 20.6415 - val_accuracy: 0.0112  
Epoch 4/5  
620/620 [=====] - 21s 34ms/step - loss: 0.5420 - accuracy: 0.8144 - val_loss: 21.6240 - val_accuracy: 0.0094  
Epoch 5/5  
620/620 [=====] - 21s 34ms/step - loss: 0.4631 - accuracy: 0.8414 - val_loss: 21.1375 - val_accuracy: 0.0125
```

Figure 18: training the model

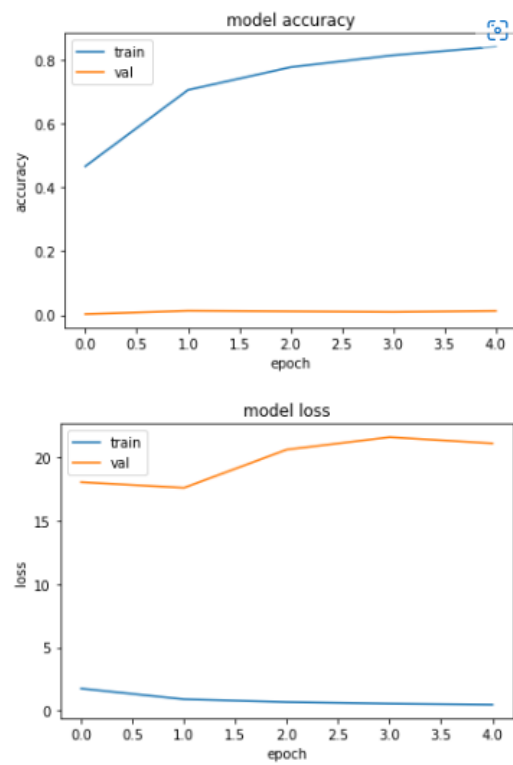


Figure 19: plots for model accuracy and loss

The validation accuracy of our model is significantly lower than the training accuracy, this signifies over-fitting which is then attempted to be solved through three different approaches, all to no avail.

8.2 Adding drop out layer

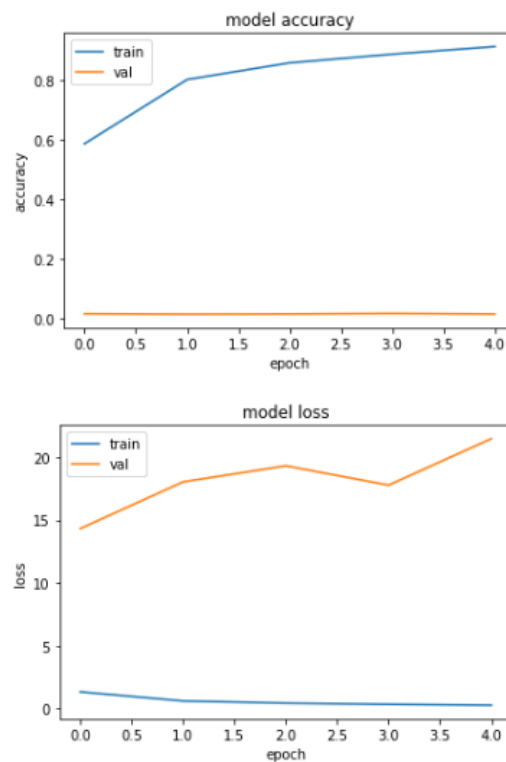


Figure 20: plots for model accuracy and loss after adding dropout layer(s)

8.3 Reducing convolution layers

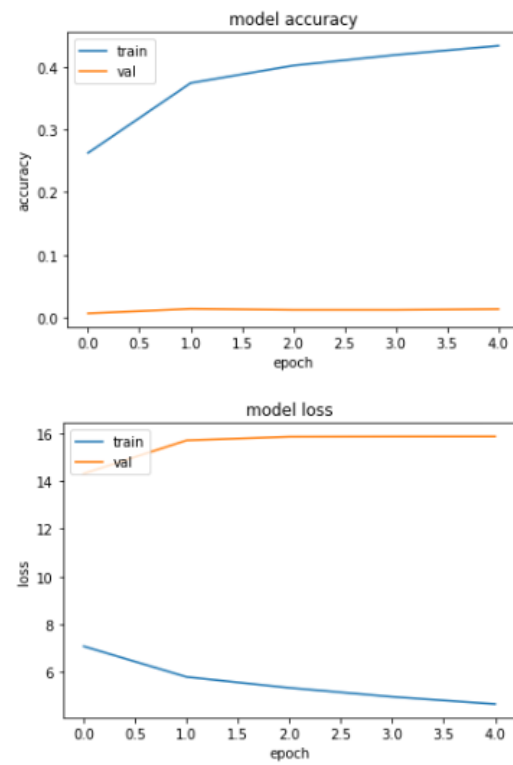


Figure 21: plots for model accuracy and loss after both reducing convolution layers and adding dropout layer

8.4 Early stopping

```
: #trying early stopping
from tensorflow import keras
es_callback = keras.callbacks.EarlyStopping(monitor='val_loss', patience=3)
history= model.fit(x_train_resized, y_train, epochs=5,validation_split=0.3,callbacks=[es_callback]) #training model
```

Epoch 1/5
620/620 [=====] - 15s 24ms/step - loss: 4.4680 - accuracy: 0.4464 - val_loss: 15.8420 - val_accuracy: 0.0151
Epoch 2/5
620/620 [=====] - 15s 24ms/step - loss: 4.2991 - accuracy: 0.4654 - val_loss: 15.8534 - val_accuracy: 0.0142
Epoch 3/5
620/620 [=====] - 15s 24ms/step - loss: 4.2032 - accuracy: 0.4610 - val_loss: 15.8552 - val_accuracy: 0.0139
Epoch 4/5
620/620 [=====] - 15s 24ms/step - loss: 4.0616 - accuracy: 0.4704 - val_loss: 15.8610 - val_accuracy: 0.0139

Figure 22: early stopping

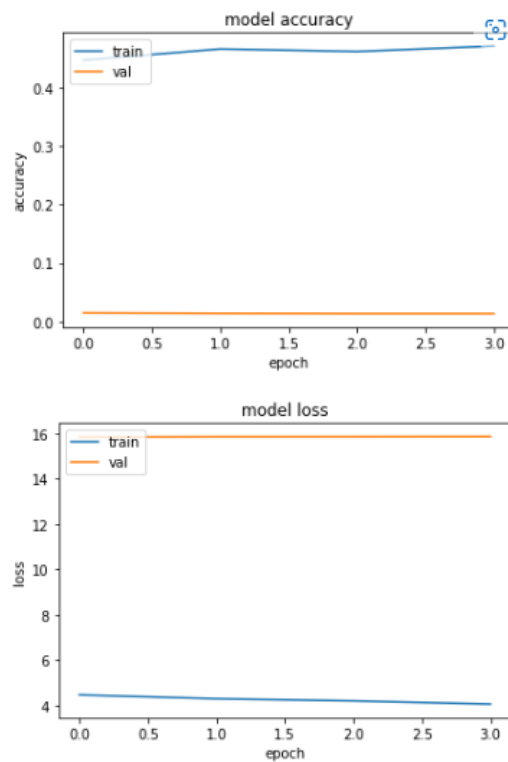


Figure 23: plots after trying early stopping

9 Evaluating model

The accuracy of our model is not very satisfactory. This is likely due to the unfruitful endeavors of avoiding or reducing the over-fitting issue.

```
: #evaluating on test data

test_loss, test_acc = model.evaluate(x_test_resized,y_test)
print('test loss on test samples: ', test_loss)
print('validation accuracy on test samples: ', test_acc)

153/153 [=====] - 2s 11ms/step - loss: 3575.8362 - accuracy: 0.4658
test loss on test samples: 3575.836181640625
validation accuracy on test samples: 0.4657786786556244
```

Figure 24: model evaluation on test data

9.1 Predicting on test data

```
: predictions = model.predict([x_test_resized])

153/153 [=====] - 2s 9ms/step
```

Figure 25: running model to predict on test data

```
print(predictions)
[[1. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```

```
print(np.argmax(predictions[0]))
```

```
0
```

```
plt.imshow(x_test[0])
```

```
<matplotlib.image.AxesImage at 0x2ac54f50130>
```

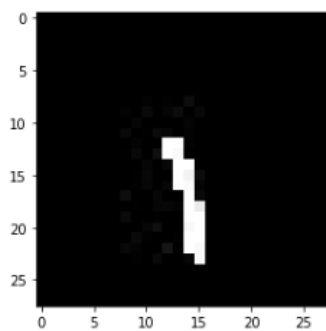


Figure 26: predicting on test data

```
: print(np.argmax(predictions[650]))
```

```
4
```

```
: plt.imshow(x_test[650])
```

```
: <matplotlib.image.AxesImage at 0x2ac4d1f25e0>
```

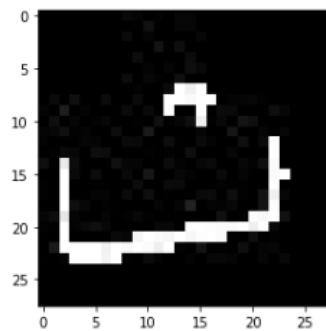


Figure 27: predicting on test data


```
romanNames[y_test[650]]
```

```
'tey'
```

Figure 28: predicting on test data

9.2 predicting on imported images

We drew an image on Paint. That picture was then read using OpenCV, did pre-processing on the image and ran our model on it to get a prediction.

```
: img3=cv2.imread('bey.png')
plt.imshow(img3)
gray3 = cv2.cvtColor(img3, cv2.COLOR_BGR2GRAY)
resized3 = cv2.resize(gray3,(28,28),interpolation=cv2.INTER_AREA)
newimg3=tf.keras.utils.normalize(resized3, axis=1)
newimg3 = np.array(newimg3).reshape(-1,img_size,img_size,1)
```

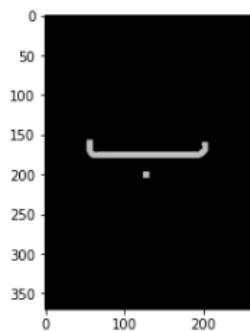


Figure 29: reading and pre-processing paint drawn character

```
predictions = model.predict(newimg3)

1/1 [=====] - 0s 26ms/step

pred = np.argmax(predictions)
print('class: ', pred)
print('character: ', romanNames[pred])

class: 4
character: tey
```

Figure 30: prediction on the read image

10 Graphical User Interface (GUI)

We used python flask to build a simple GUI for our project, the trained model was saved and then later loaded and used in the flask file, Our web interface clicks the picture when the submit button is pressed and CV2 is used to process the image to be sent to the trained model for classification

```
C:\Users\Abdul Basit\Downloads\Wev Page\app.py
temp.py x app.py x
1  from flask import Flask, render_template, Response, request
2  import cv2
3  import numpy as np
4  import tensorflow as tf
5
6  app = Flask(__name__)
7  video = cv2.VideoCapture(0)
8
9  @app.route('/')
10 def index():
11     """Video streaming home page."""
12     return render_template('index.html')
13
14 @app.route('/takeimage', methods = ['POST','GET'])
15 def takeimage():
16     name = request.form['name']
17     # print(name)
18     _, img = video.read()
19
20
21
22     model = tf.keras.models.load_model("/tmp/model")
23     model.summary()
24
25
26     romanNames=['alif','alif mad aa','bey','pey','tey','ttey',
27                 'sey','jeem','chay','bari he','khe','daal','dhaal','zaal',
28                 're','rhey','zaa','zhaa','seen','sheen','svaad','zvaad',
29                 'toy','zoy','ain','ghain','fey','qaaf','kaaf','gaaf','Laam',
30                 'meem','noon','noonghunna','wao','choti he','do chashmi he',
31                 'ye','bari ye']
32
33
34
35     gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
36     resized = cv2.resize(gray,(28,28),interpolation=cv2.INTER_AREA)
```

Figure 31: Code of python flask

11 References

- [1] Recognition of Urdu Handwritten Characters Using Convolutional Neural Network Mujtaba Husnain , Malik Muhammad Saad Missen , Shahzad Mumtaz ,Muhammad Zeeshan Jhanidr, Mickaël Coustaty, Muhammad Muzzamil Luqman, Jean-Marc Ogier and Gyu Sang Choi
- [2] Urdu Optical Character Recognition Systems NAILA HABIB KHAN AND AWAIIS ADNAN
- [3] Urdu handwritten text recognition Mujtaba Husnain,Malik Muhammad Saad Missen,Shahzad Mumtaz,Mickaël Coustaty,Muzzamil Luqman,Jean-Marc Ogier

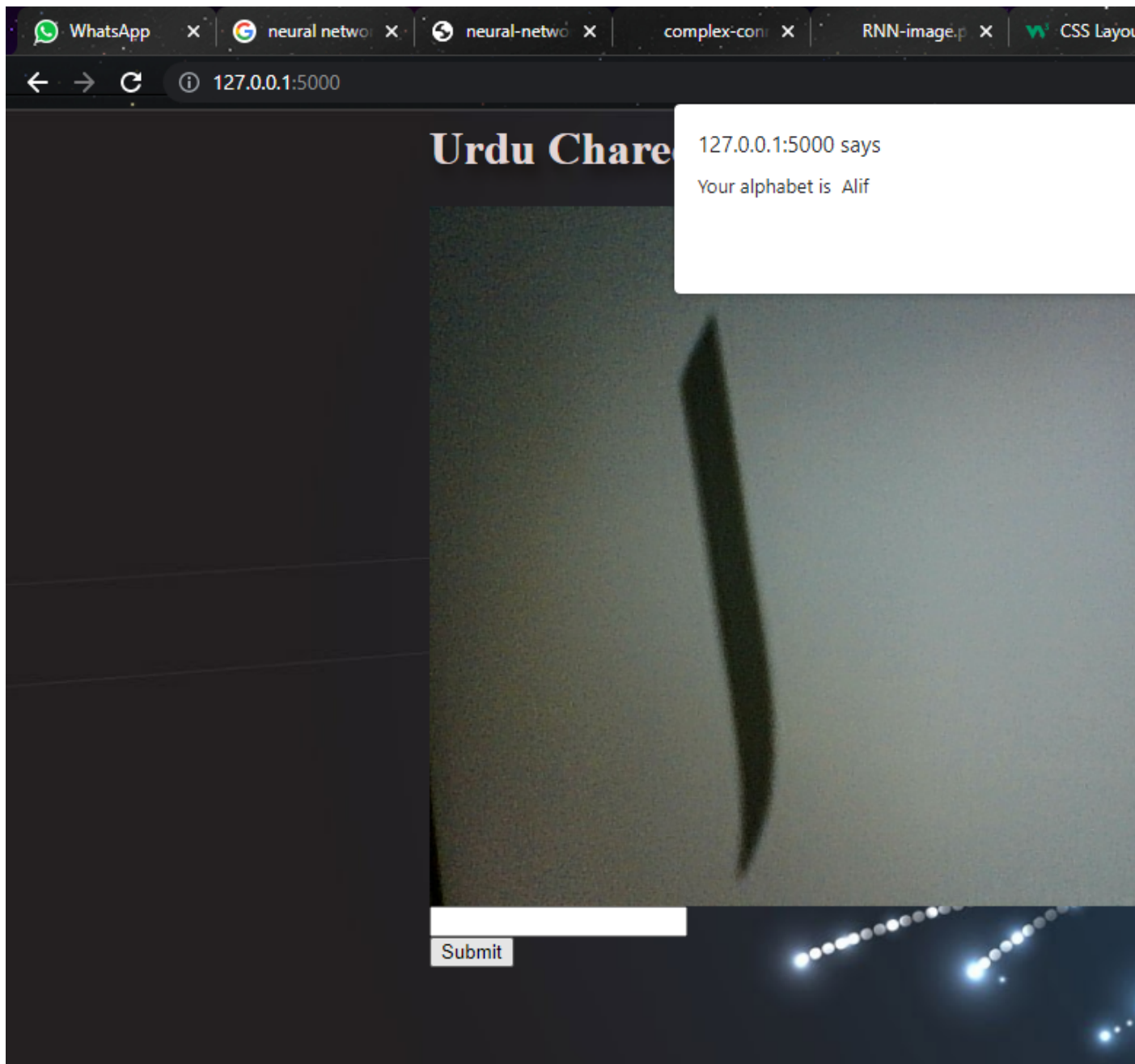


Figure 32: Prediction of the letter