Project Report

on

Writer Verification on Multi-Language Script using Deep Learning

(A dissertation submitted in partial fulfilment of the requirements of Bachelor of Technology in Computer Science and Engineering of the Maulana Abul Kalam Azad University of Technology, West Bengal)

Submitted by

Souporno Ghosh
Soumya Nasipuri
Rahul Roy
Sharanya Saha

Under the guidance of
Prof. Jaya Paul

Asst. Prof.,
Dept. of Computer Science and Engineering

Government College of Engineering and Leather Technology

(Affiliated to MAKAUT, West Bengal)

Kolkata - 700106, WB

2020-2021

Certificate of Approval

This is to certify that the project report on "Writer Verification on Multi-Language Script using Deep Learning" is a record of bona fide work, carried out by Shri Souporno Ghosh, Shri Soumya Nasipuri, Shri Rahul Roy and Smt Sharanya Saha under my guidance and supervision.

In my opinion, the report in its present form is in conformity as specified by Government College of Engineering and Leather Technology and as per regulations of the Maulana Abul Kalam Azad University of Technology, West Bengal. To the best of my knowledge the results presented here are original in nature and worthy of incorporation in project report for the B.Tech. Program in Computer Science and Engineering.

Signature of
Supervisor/ Guide

Signature of
Head, Dept. of CSE

ACKNOWLEDGEMENT

With great pleasure, I would like to express my profound gratitude and indebtedness to Prof Jaya Paul, Department of Computer Science and Engineering, Government College of Engineering and Leather Technology, W.B. for her continuous guidance, valuable advice and constant encouragement throughout the project work. Her valuable and constructive suggestions at many difficult situations are immensely acknowledged. I am in short of words to express her contribution to this thesis through criticism, suggestions and discussions.

I would like to take this opportunity to thank Prof Jaya Paul, Project Coordinator and Dr. Santanu Halder, HOD, Department of Computer Science & Engineering, Government College of Engineering and Leather Technology.

I would like to express my gratitude to Sri S. Mondal for his valuable suggestions and help.

Signatures

1. Souporno Ghosh - 11200117028
2. Soumya Nasipuri - 11200117029
3. Rahul Roy - 11200117039
4. Sharanya Saha - 11200117033

Dedicated to

Alan Turing, Ada Lovelace and John von Neumann

The pioneers on whose work we expand upon.

ABSTRACT

Significant amount of research has been done in the field of handwriting recognition, particularly for characters in the Latin-based alphabets (English, French, Spanish, German, etc). However, there is a significant shortage of literature and research on handwriting recognition for Devanagari based languages, such as Hindi, Bangla, Sanskrit, etc. In this project, we attempt to remedy that in an attempt to create an API that is able to recognize the writer for a passage written in Bangla handwriting. The primary goal of the API is to identify the author of a word, sentence or passage from the handwriting written in Bangla. We also provide the methods used by us in this attempt in order to facilitate further study and replication of this API for future research. In this project, we will apply the VGG16 model to identify the writer of a text written in Bangla. We will also compare our accuracy with previous works in this field.

CONTENTS

CHAPTER 1. INTRODUCTION

1. Motivation

Our great nation has produced many literary geniuses; Munshi Prem Chand, Rabindranath Tagore, Vikram Seth, Bankim Chandra Chatterjee, Sukumar Roy, etc. These pioneers have blessed us with a variety of literary masterpieces that provide not only an insight into their own minds but also insight on humanity and contemporary times. It is sufficed to say one learns a lot about mankind from their works. Moreover, their works and by extension, the manuscripts of said works, are a national treasure. Hence, it is imperative that their original works can be verified as their own. We can achieve this by analysing the handwritings of the writers.

Handwriting recognition will also help accelerate the field of forensic analysis, and in turn help the law enforcement authorities. Notes found on crime scenes and related to victims and suspects can be analysed and such analysis can help us identify perpetrators of a crime.

the field of education and academia, handwriting analysis can help us curb plagiarism. Plagiarism checking is a major field of research in academia and handwriting analysis can also help support those efforts.

These are only a few of the applications of handwriting recognition that we drove us to choose this topic for our project. Recognition and analysis of handwriting has applications in various fields such as archaeology, criminal detection, academia, education, etc. However, so far handwriting analysis has only been performed by human hands. In modern days, handwriting recognition has mostly only been attempted for languages based on Latin-based alphabet.

Literature related to handwriting recognition is limited for Devanagari related languages, such as Hindi and Bangla. Hence, our feeble attempt at remedying that.

2. Background

To decide what approach, we should take to recognise the writer from the handwriting, we decided to survey related literature. First, we look into image recognition basics to find out the best way to approach the image recognition problem. We looked into the validity of Deep Convolutional Neural Networks for image recognition as discussed by Krizhevsky et al. [1]. Additionally, we referenced the original article [2] where the VGG16 was first proposed.

To understand how handwriting can be treated as images, we looked into articles on handwritten character recognition. We referred to articles by Rehman et al. [3], John et al. [4], Hasnat et al. [5] and Paul et al. [6, 7]. Hasnat et al. proposed a domain specific OCR which classify machine printed as well as handwritten Bangla characters. For feature extraction they apply Discrete Cosine Transform (DCT) technique over the input image and for classification Hidden Markov Model (HMM) was used [5]. Paul et al. has attempted Bangla character recognition with Mobilenet v1 and Inception v3 [6] and Bangla number recognition with Convolutional Neural Networks [7]. While the works are different from what we are trying achieve, they help us find an appropriate way to approach handwriting.

For author recognition we looked into the works of Christlein et al. [8], Schlapbache et al. [9] and Wu et al. [10]. Christlein et al. [8] attempted author recognition with Convolutional Neural Network. Schlapbache et al. analysed HMM based handwriting recognition systems and studied the effect of normalisation operations [9]. Wu et al. attempted writer identification on English and Chinese languages [10].

3. Summary of present work

As mentioned earlier, the work done in the field of Bangla character recognition is very limited. For reference and guidance, we looked into the work of Adak et al. [11].

In the article, high intra-variable handwriting-based writer identification/verification is attempted. Both handcrafted and auto-derived feature-based models are considered to study writer identification/verification performance.

Two offline Bangla intra-variable handwriting databases from two different sets of 100 writers are used; Controlled (D_C) and Uncontrolled (D_{UC}). For writer identification, multi-class classification was used (the

number of classes is equal to the total count of writers) where the task was to assign the writer-id to the unknown handwritten specimens. For writer verification, a binary classification was used where the task is to answer “yes” or “no” to the handwritten sample in question.

Both writer identification and writer verification were tried out with two methods: Handcrafted Feature-Based Identification and Auto-Derived Feature-Based Identification. The process used for Handcrafter Feature extraction are: Macro-Micro Features (F_{MM}), Contour Direction and Hinge Features (F_{DH}) and Direction and Curvature Features at Key points (F_{DC}). For Auto-Derived Feature Extraction Basic_CNN, SqueezeNet, GoogLeNet, Xception Net, VGG16 and ResNet 101 are used.

After experimentation on the databases, it is observed that by training and testing on similar writing variability, the system produces encouraging outcomes. However, the system performance is comparatively lower for training and testing on disparate types of handwriting variability. Cross learning is also attempted and it is observed that the system performance improves with pre-training. Here, a practical scenario is imitated, whereby a certain writing style of an individual is unknown (i.e., absent during training), and we note that the state-of-the-art methods do not perform well.

The following tables contains the summarised result of their work.

The accuracies corresponding to writer verification on an enlarged set is shown below. Since, we have attempted writer verification with VGG16 model in our project, we have only mentioned the verification accuracy for VGG16 model as measured by Adak et al. here.

The following is the table containing accuracies when the model was trained with and tested on a controlled database.

Model	Accuracy (%)
VGG16 on Character Level Features	97.77
VGG16 on Stroke Level Features	97.20

Table 1.1. Accuracy for Writer Verification on Controlled Data Set

The following is the table containing accuracies when the model was trained with and tested on an uncontrolled database.

Model	Accuracy (%)
VGG16 on Character Level Features	97.36
VGG16 on Stroke Level Features	96.43

Table 1.2. Accuracy for Writer Verification on Uncontrolled Data Set

4. Organisation of the Thesis

The thesis has been organised into primarily four chapters.

The Data Collection and Pre-processing chapter deals with the details of all the work that must be done before the actual programming part can be approached. This includes, but is not limited to data collection and preparation

The Methodology chapter deals with the necessary theory required to approach the problems. This includes any experiments we ran to test out our algorithms and any preliminary work which we referred to help with our project.

The Program Walkthrough chapter contains a step-by-step analysis of the actual program written for the image preparation and the creation of the model.

The Outputs and Results chapter deals with the output that we get from executing the program and how they are significant.

5. Hardware/Software used

Primary language used for programming is Python 3.8.

Packages used in Python are TensorFlow v2.0 (a deep learning framework by Google, Inc.), Keras (to help with integrating TensorFlow with Python), NumPy (NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays), Pandas (software library written for the Python programming language for data manipulation and analysis), Matplotlib (used for plotting data and graphs) and Seaborn (Python data visualization library based on matplotlib; It provides a high-level interface for drawing attractive and informative statistical graphics).

The primary algorithm used for image (handwriting) recognition is VGG16.

The primary hardware used comprise the personal laptop computers belonging to the team members; HP Pavilion with 2.7GHz Quad-Core Intel i5, Integrated Graphics Card, 8GB RAM; Dell G3 with 2.6GHz Hexa-Core Intel i7 Processor, Integrated Graphics Card, 8GB RAM and Asus Vivobook 2GHz Quad-Core AMD Ryzen 5 Processor, Integrated Graphics Card, 8GB RAM.

The processor and the RAM determine how fast the program will run and the dataset will be trained. The graphics processing unit would also have accelerated the process, if GPU parallel processing was implemented in the project. A major issue with deep learning projects is that if the processing power is too less, then the model might take as long as a few days to train. However, our initial project does not take as much time and can be replicated with a machine with as low specifications as 1.6 GHz Dual Core Processor, Integrated Graphics Card and 4GB RAM. Anything lower might take too long to process. (Training the Dataset will require much better specifications, however, since the original VGG16 model was trained on 2-3 weeks on a Nvidia Titan GPU [2]. In such a scenario, training the data on a cloud platform like GCP, AWS or Azure might yield better results).

CHAPTER 2. INITIAL STAGES

1. Collection of Dataset

A major criterion of success in working on problems whose solutions depend on machine or deep learning is the presence of large datasets. The larger the dataset, the better and more accurate the model becomes.

One of the main challenges of attempting handwriting recognition in a new language is the lack of sufficient datasets. Our supervisor for the project, Smt Jaya Paul, was generous enough to assist us with that. We used 121 volunteers of various native languages and asked them to write certain passages in their native and English languages. This provided us with the necessary dataset needed for proper model training.

For our project, we have used the dataset containing Bangla words only.

2. Preparation of Dataset

The handwritten passages were scanned into image. The scanned images passages were processed to eliminate noise and then segmented into word-sized images. The methods used for this was provided to us from a previous project attempted by Mondal et al. [12]. These word sized images are then arranged into several folders in the following method.

- The images are distinguished according to pairs of authors and for each author, there are five (5) sets of data.
- Each of these 5 sets are divided into two subfolders; train containing three sets and test containing two sets.
- The images are in Tag Image File Format (TIFF). They are named in the following format: <Author Code>_<Set Number>_<Image Number>. For example, the first image of the first set of the first author (Author Code 0) is *0000_01_0.tiff*.
- There are five thusly organized datasets.

These images are now ready to be fed as inputs for the model.

3. Construction of the Model

Currently, the base model that we will be using is called VGG16. VGG16 is one of the most popular models for image recognition. It was introduced in 2014 [2]. VGG16 takes the input in 224 x 224 images and the pretrained VGG 16 model (with the ImageNet Database) will predict from the 1000 categories provided in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2014.

In our dataset, the size of the segmented images is random, and hence some pre-processing is required so that the input images can be converted to the required size of 224 x 224. We shall do that with the help of the in-built VGG 16 pre-processing tools provided in the Keras library. The Program Walkthrough mentions exactly how we do it.

CHAPTER 3. METHODOLOGY

1. About VGG 16 Model [2]

VGG 16 was proposed by Karen Simonyan and Andrew Zisserman of the Visual Geometry Group Lab at the University of Oxford in 2014 in an article titled "Very Deep Networks for Large Scale Image Recognition." This model achieves 92.7% on the top 5 accuracy tests with the ImageNet dataset of 14 million images belonging to 1000 categories. The model inputs 224 x 244 pixels size images in RBG channels.

1.1 Architecture of VGG16:

The input to the network is image of dimensions (224, 224, 3). The first two layers have 64 channels of 3*3 filter size and same padding. Then after a max pool layer of stride (2, 2), two layers which have convolution layers of 256 filter size and filter size (3, 3). This followed by a max pooling layer of stride (2, 2) which is same as previous layer. Then there are 2 convolution layers of filter size (3, 3) and 256 filter. After that there are 2 sets of 3 convolution layer and a max pool layer. Each have 512 filters of (3, 3) size with same padding. This image is then passed to the stack of two convolution layers. In these convolution and max pooling layers, the filters we use is of the size 3*3 instead of 11*11 in Alex Net and 7*7 in ZF-Net. In some of the layers, it also uses 1*1 pixel which is used to manipulate the number of input channels. There is a padding of 1-pixel (same padding) done after each convolution layer to prevent the spatial feature of the image.

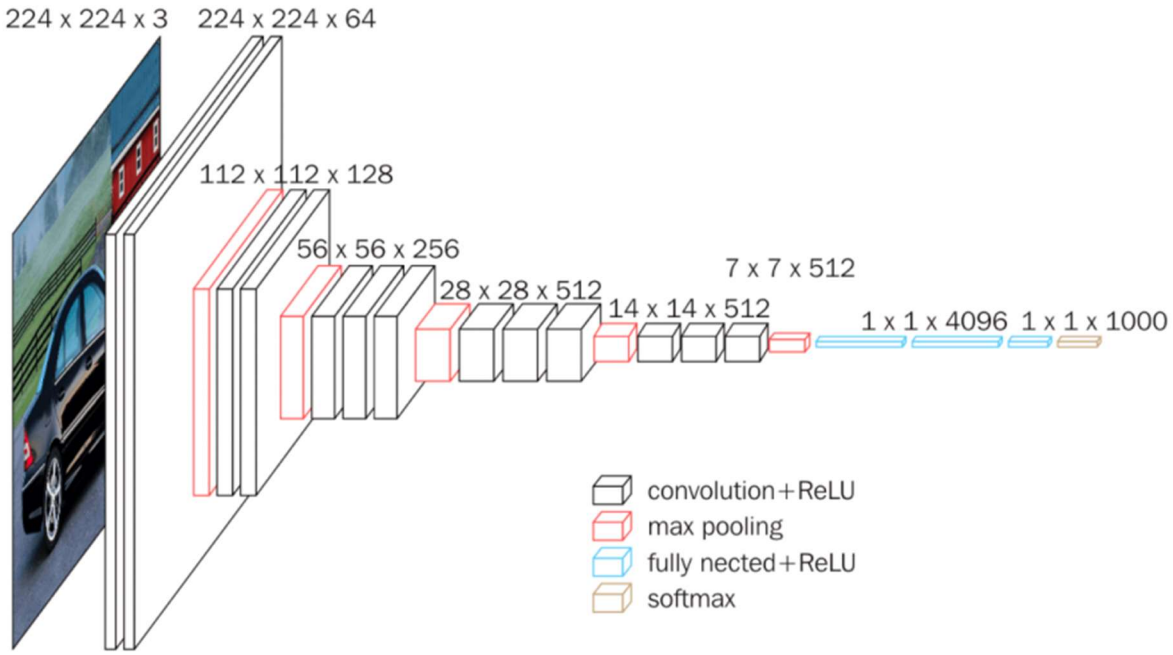

Fig 3.1. Architecture of VGG16 Model

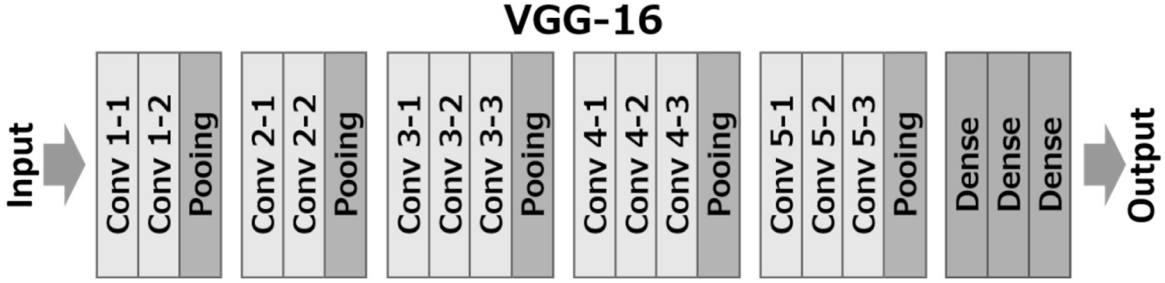

Fig 3.2. Flowchart of VGG16 Model

After the stack of convolution and max-pooling layer, we got a (7, 7, 512) feature map. We flatten this output to make it a (1, 25088) feature vector. After this there are 3 fully connected layer, the first layer takes input from the last feature vector and outputs a (1, 4096) vector, second layer also outputs a vector of size (1, 4096) but the third layer output 1000 channels for 1000 classes of ILSVRC challenge, then after the output of 3rd fully connected layer is passed to SoftMax layer in order to normalize the classification vector. After the output of classification vector top-5 categories for evaluation. All the hidden layers use ReLU as its activation function. ReLU is more computationally efficient because it results in faster learning and it also decreases the likelihood of vanishing gradient problem.

This is how the pretrained VGG16 model works.

1.2 Challenges of VGG16:

- It is very slow to train (the original VGG model was trained on NVidia Titan GPU for 2-3 weeks).
- The size of VGG16 trained ImageNet weights is 528 MB. So, it takes quite a lot of disk space and bandwidth that makes it inefficient.

2. Why VGG16?

Recognizing images has always been more efficient with the help of deep neural networks. A. Krizhevsky, I. Sutskever and G.E. Hinton [1] have shown that while using a huge dataset like ImageNet, the results show that a large, deep convolutional neural network is capable of achieving record-breaking results on a highly challenging dataset using purely supervised learning. It is notable that our network's performance degrades if a single convolutional layer is removed.

VGG16 happens to be one of the best models for image recognition with accuracy 92.7% on the top 5 accuracy tests on the ImageNet dataset of 14 million images belonging to 1000 categories [2]. Handwriting can be considered as a pattern to be recognised. Hence, we decided that VGG16 model might be the best path to proceed with the handwriting recognition.

3. Initial Experimentation

Before we proceeded to handwriting recognition, we experimented with a cat and dog classifier [13]. The dataset we found was from Kaggle [14]. We attempted to create our own VGG16 model in order to better understand the VGG16 model. We shall once walk through the program to understand the VGG16 model better.

First, we import Keras framework and OS library. Keras is the package that helps us use the TensorFlow 2.0 Library by Google, Inc.

```
import keras
import os
```

Then we import the necessary modules from keras. This include but are not limited various algorithms that will constitute each layer of the model (namely Dense, Convolution, Max Pool and Flattening), image pre-processing tools and various keras utilities.

```
from keras.models import Sequential
from keras.layers import Dense, Conv2D, MaxPool2D, Flatten
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ModelCheckpoint, EarlyStopping
from keras.preprocessing import image
from keras.models import load_model
from keras.optimizers import Adam
```

We will also import NumPy and Mat Plot Library. NumPy assists us with handling matrices and Mat Plot Library helps us to view graphs and plots.

```
import numpy as np
import matplotlib.pyplot as plt
```

Here we will load the datasets into the program. The data is divided into two subsets; test and train.

```
trdata = ImageDataGenerator()
traindata = trdata.flow_from_directory(
    directory="./train", target_size=(224, 224))
tsdata = ImageDataGenerator()
testdata = tsdata.flow_from_directory(
    directory="./test", target_size=(224, 224))
```

From here we will add will add all the layers relevant to the VGG16 model in sequentially.

```
model = Sequential()
```

The following lines of code will add the first set of layers. This includes two Convolutional networks. The first one takes in the image of size 224 x 224 in RGB channels. The next one has 64 filters. ReLU is often used as the activation function. The output after these two layers is pooled with max pooling algorithm.

```
model.add(Conv2D(input_shape=(224, 224, 3), filters=64,
    kernel_size=(3, 3), padding="same", activation="relu"))
model.add(Conv2D(filters=64, kernel_size=(3, 3),
    padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2, 2), strides=(2, 2)))
```

Then we will add two more convolutional networks with 128 filters, activation function ReLU and the same padding. The output after these two layers is pooled with max pooling algorithm.

```
model.add(Conv2D(filters=128, kernel_size=(
    3, 3), padding="same", activation="relu"))
model.add(Conv2D(filters=128, kernel_size=(
    3, 3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2, 2), strides=(2, 2)))
```

Then we will add three more convolutional networks with 256 filters, activation function ReLU and the same padding. The output after these two layers is pooled with max pooling algorithm.

```
model.add(Conv2D(filters=256, kernel_size=(
    3, 3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(
    3, 3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(
    3, 3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2, 2), strides=(2, 2)))
```

Then we will add three more convolutional networks with 512 filters, activation function ReLU and the same padding. The output after these two layers is pooled with max pooling algorithm.

```
model.add(Conv2D(filters=512, kernel_size=(
    3, 3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(
    3, 3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(
    3, 3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2, 2), strides=(2, 2)))
```

Then we will add another set of three more convolutional networks with 512 filters, activation function ReLU and the same padding. The output after these two layers is pooled with max pooling algorithm.

```
model.add(Conv2D(filters=512, kernel_size=(
    3, 3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(
    3, 3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(
    3, 3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2, 2), strides=(2, 2)))
```

We use Rectified Linear Unit (ReLU) for activation so that the negative values are not passed to the next layers. After this, we will add two Dense layers with ReLU activation with 4096 units followed by a SoftMax Dense layer with two units.

```
model.add(Flatten())
model.add(Dense(units=4096, activation="relu"))
model.add(Dense(units=4096, activation="relu"))
model.add(Dense(units=2, activation="softmax"))
```

Here, we will be using Adam optimiser to reach to the global minima while training out model. If the algorithm stuck in local minima while training then the Adam optimiser will help us to get out of local minima and reach global minima. We will also specify the learning rate of the optimiser, here in this case it is set at 0.001. If our training is bouncing a lot on epochs then we need to decrease the learning rate so that we can reach global minima.

```
opt = Adam(lr=0.001)
model.compile(optimizer=opt,
        loss=keras.losses.categorical_crossentropy, metrics=['accuracy'])
```

We will then print the summary of the model.

```
model.summary()
```

We shall use the ModelCheckpoint Utility from Keras library to save the model only when the validation accuracy of the model in current epoch is greater than what it was in the last epoch. The EarlyStopping Utility from Keras library to stop the learning in case there is no increase in validation accuracy for 20 epochs.

```
checkpoint = ModelCheckpoint("vgg16_1.h5", monitor='val_acc', verbose=1,
                        save_best_only=True, save_weights_only=False, mode='auto',
period=1)
early = EarlyStopping(monitor='val_acc', min_delta=0,
                        patience=20, verbose=1, mode='auto')
hist = model.fit_generator(steps_per_epoch=100, generator=traindata,
validation_data=testdata,
                        validation_steps=10, epochs=100, callbacks=[checkpoint, early])
```

The fit generator method in the model class helps us pass the data to the model for training and validation. The method handles everything necessary for passing data to the model and training the model. After this we will print the information about the model training with the help of the MatPlot Library.

```
plt.plot(hist.history["acc"])
plt.plot(hist.history['val_acc'])
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title("model accuracy")
plt.ylabel("Accuracy")
plt.xlabel("Epoch")
plt.legend(["Accuracy", "Validation Accuracy", "loss", "Validation Loss"])
plt.show()
```

Now to predict using the model that we created, we will import an image and use the Keras library to preprocess it for the VGG16 model we created.

```
img = image.load_img("image.jpeg", target_size=(224, 224))
img = np.asarray(img)
plt.imshow(img)
img = np.expand_dims(img, axis=0)

saved_model = load_model("vgg16_1.h5")

output = saved_model.predict(img)

if output[0][0] > output[0][1]:
    print("cat")
else:
    print('dog')
```

This above experimental program gives us in depth understanding of the VGG16 algorithm and its architecture.

4. Extracting Features

Before we could start training the actual model, we needed to pre-process the input. This is primarily because the input for the VGG16 model requires the images to be of size 224 x 224 pixels and RGB channel. Additionally, before we train the model, we will extract the necessary feature required for prediction.

To obtain the necessary features, we take the pretrained VGG16 model available in the Keras library and remove the top layer, where the prediction (with the 1000 categories from ILSVRC challenge) actually happens. Then we pass the data through it to get a "prediction" to get a feature matrix as an output. The below program does that for one image in the dataset.

First, we will import all the necessary packages. This includes Keras library, NumPy, Pandas, Mat Plot Library and Seaborn. We also import all the necessary utilities from the keras library. This includes the VGG16 model that has been pretrained with the ImageNet Dataset, image preprocessing tools and image data generator.

```
import keras
from PIL import Image
from keras.preprocessing import image
from keras.applications.vgg16 import VGG16
from keras.applications.vgg16 import preprocess_input
from keras.preprocessing.image import ImageDataGenerator

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

sns.set()
```

This line initialises seaborn.

From the existing pretrained model, we will remove the top layer. The top layer is responsible for actually predicting the category of the image and it does that from the 1000 categories provided by the 1000 categories from ILSVRC challenge. If this layer is removed, then the classification does not occur. Instead, we get the features necessary for the classification of the images. The weights of this pretrained model are configured according to the ImageNet dataset.

```
model = VGG16(weights='imagenet', include_top=False)

model.summary()
```

The image_loader method will pre-process the images and create the necessary data from the image that can be used as an input to the VGG16 model to obtain the necessary feature matrix.

```
def image_loader(img_path):

    img = image.load_img(img_path, target_size=(224, 224))
    plt.imshow(img)

    img_data = image.img_to_array(img)
    print(img_data.shape)

    img_data = np.expand_dims(img_data, axis=0) # google axis = 0
    print(img_data.shape)

    img_data = preprocess_input(img_data)
    print(img_data.shape)

    return img_data

img_path = '../Datasets/writerPair1/0-10/train/0/0000_01_0.tif'
img_data = image_loader(img_path)
```

Now we will use the data we get from the image loader function to obtain the feature matrix. Here, we obtain the feature matrix for one image.

```
model_feature = model.predict(img_data)

print(model_feature.shape)
print(feature_matrix.shape)

df = pd.DataFrame(feature_matrix)
df.to_csv('feature.csv',index=False)
```

In the actual program, we will be looping through all the images so that we can get the features for all the images. We will store the feature matrices in CSV files. We will create one CSV file for each set of data. Since there are 5 sets of data for each author pair (3 for training and 2 for verification), we will have 5 CSV files containing the features of all the images. We will use these feature matrices to train and validate our model.

5. Extracted Features

We will input the following image to the updated (i.e. pretrained VGG16 without the top layer) model.

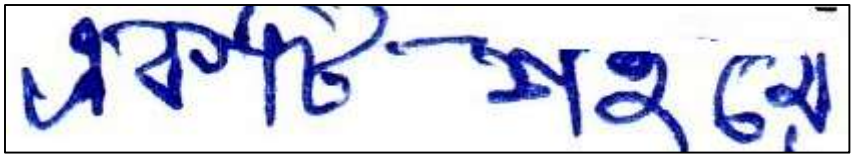

Fig 3.3. Segmented image input to the updated model

Running the earlier program gives us the following output.

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, None, None, 3)]   0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, None, None, 64)    1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, None, None, 64)    36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, None, None, 64)    0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, None, None, 128)   73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, None, None, 128)   147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, None, None, 128)   0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, None, None, 256)   295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, None, None, 256)   590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, None, None, 256)   590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, None, None, 256)   0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, None, None, 512)   1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, None, None, 512)   2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, None, None, 512)   2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, None, None, 512)   0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, None, None, 512)   2359808
```

```
_________________________________________________________________
block5_conv2 (Conv2D)        (None, None, None, 512)   2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, None, None, 512)   2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, None, None, 512)   0
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
(224, 224, 3)
(1, 224, 224, 3)
(1, 224, 224, 3)
2021-03-27 23:23:36.385607: I
tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:116] None of the MLIR
optimization passes are enabled (registered 2)
(1, 7, 7, 512)
(25088,)
```

The following is part of the feature matrix (NumPy does not print all of the values if the array is too big). It is a 1 x 7 x 7 x 512 matrix. Our goal will be appending the matrices for all the sets to obtain the features relevant to an author and then use the features to verify the handwriting with respect to the writer.

```
[[[[ 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.         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.         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.        ...  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.         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.
   0.        ]
 ...
 [ 0.          0.          0.          ...  0.          0.9801242
   0.        ]
 [ 0.          0.          0.          ...  0.          0.
   0.        ]
 [ 0.          0.          0.          ...  0.          0.
   0.        ]]

[[ 0.          0.          0.          ...  0.          1.8023927
   0.        ]
 [ 0.          0.          0.          ...  0.          0.
   0.        ]
 [ 0.          0.          0.          ...  0.          0.
   0.        ]
 ...
 [ 0.          0.          0.          ...  0.          6.553742
   0.        ]
 [ 0.          0.          0.          ...  0.          0.
   0.        ]
 [ 0.          0.          0.          ...  0.          0.
   0.        ]]

[[ 0.          0.          0.          ...  0.          6.669153
   0.        ]
 [ 0.          0.          0.          ...  0.          3.7987955
   0.        ]
 [ 0.          0.          0.          ...  0.          0.
   0.        ]
 ...
 [ 0.          0.          0.          ...  0.         14.567184
   0.        ]
 [ 0.          0.          0.          ...  0.          4.3287916
   0.        ]
 [ 0.          0.          0.          ...  0.          0.
   0.        ]]]]
```

We will flatten this to create a feature matrix for a set for particular author. The resultant matrix is 25088 x 1. The following is the screenshot of the output CSV file after flattening the 1 x 7 x 7 x 512 feature matrix.

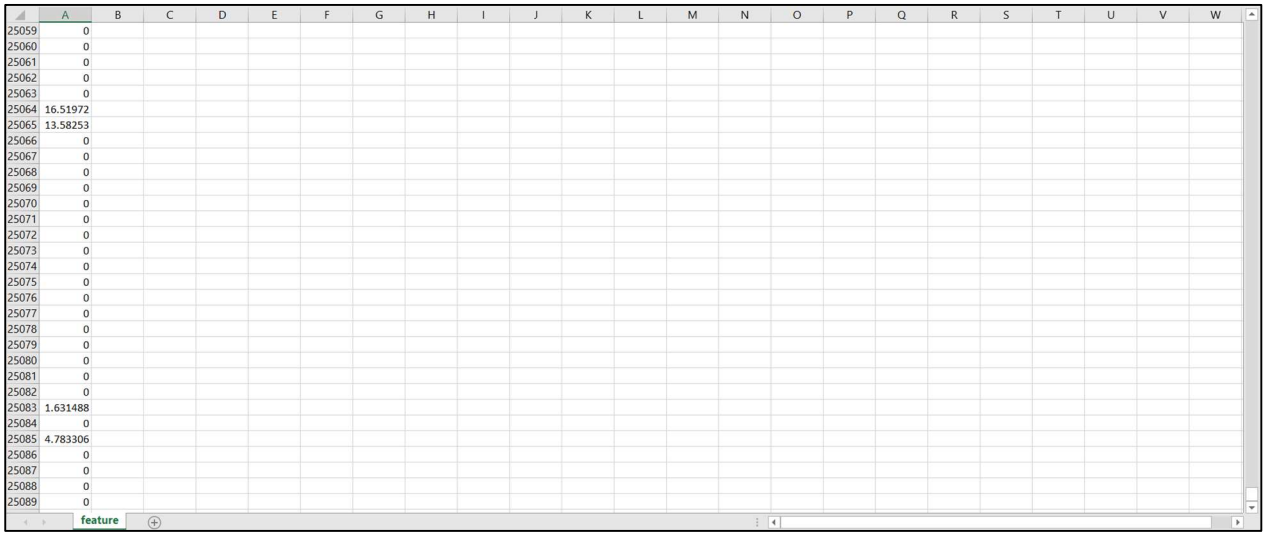

Fig 3.4. Screenshot of the CSV file with the feature matrix

CHAPTER 4. PROGRAM WALKTRHOUGH

For the actual program.

CHAPTER 5. OUTPUTS AND RESULTS

For the actual program.

CONCLUSIONS

For the actual program. Comparison between this and the actual program.

REFERENCES

[1] A. Krizhevsky, I. Sutskever and G. E. Hinton, "ImageNet classification with deep Convolutional Neural Networks", *Communications of the ACM*, Vol. 60 Issue 6, pp. 84–90, May 2017, doi: 10.1145/3065386.

[2] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition", in *Proc. International Conference on Learning Representations (ICLR 2015)*, 2015 [Online], Available: http://arxiv.org/abs/1409.1556.

[3] A. Rehman, S. Naz, M. I. Razzak and I. A. Hameed, "Automatic Visual Features for Writer Identification: A Deep Learning Approach," *IEEE Access*, Vol. 7, 2019, pp. 17149-17157, Jan. 21, 2019, doi: 10.1109/ACCESS.2018.2890810.

[4] J. John, Pramod K. V. and K. Balakrishnan, "Handwritten Character Recognition of South Indian Scripts: A Review", in *Proc. National Conference on Indian Language Computing*, Feb. 19-20, 2011.

[5] M. A. Hasnat, S. M. Habib, M. Khan, Eds., "A High Performance Domain Specific Ocr For Bangla Script", *Novel Algorithms and Techniques In Telecommunications, Automation and Industrial Electronics*, Dordrecht: Springer, 2008, doi: 10.1007/978-1-4020-8737-0_31.

[6] J. Paul, A. Roy and A. Sarkar, "Bangla character recognition based on Mobilenet v1 and Inception v3", in *Proc. International Conference on Emerging Technologies for Sustainable Development (ICETSD '19)*, Mar. 5-6, 2019, pp. 511-514.

[7] J. Paul, A. Dattachaudhuri and A. Sarkar, "CNN implementation based on Bangla numeral character recognition", in *Proc. International Conference on Emerging Technologies for Sustainable Development (ICETSD '19)*, Mar. 5-6, 2019, pp. 520-523.

[8] V. Christlein, D. Bernecker, A. Maier, and E. Angelopoulou, ''Offline writer identification using convolutional neural network activation features,'' In Proc. German Conf. Pattern Recognition, 2015, pp. 540–552. DOI:10.1007/978-3-319-24947-6_45.

[9] A. Schlapbach and H. Bunke, ''Writer identification using an HMM-based handwriting recognition system: To normalize the input or not,'' in *Proc. Conf. IGS, 2005*, pp. 138–142.

[10] X. Wu, Y. Tang and W. Bu, "Offline Text-Independent Writer Identification Based on Scale Invariant Feature Transform," in *IEEE Transactions on Information Forensics and Security*, vol. 9, no. 3, pp. 526-536, March 2014, DOI: 10.1109/TIFS.2014.2301274.

[11] C. Adak, B. B. Chaudhuri and M. Blumenstein, "An Empirical Study on Writer Identification and Verification from Intra-Variable Individual Handwriting", *IEEE Access*, Vol. 7, 2021, pp. 24738-24758, Feb 18, 2019, doi: 10.1109/ACCESS.2019.2899908.

[12] T. Mondal, S. A. Hossain, S. Mondal, R. Afroz and A. Hossain, "Preprocess the handwritten document image for preparing writer recognition", Government College of Engineering and Leather Technology, Kolkata, India, Project Report, June 2020.

[13] R. Thakur, "Step by step VGG16 implementation in Keras for beginners", *towardsdatascience.com*, Aug. 6, 2019. [Online]. Available: https://towardsdatascience.com/step-by-step-vgg16-implementation-in-keras-for-beginners-a833c686ae6c [Accessed Mar. 3, 2021].

[14] Kaggle, Inc., "Dogs vs. Cats: Create an algorithm to distinguish dogs from cats", Kaggle, Inc., Available: https://www.kaggle.com/c/dogs-vs-cats/data.