Semester Project

Write your name
1. Prathyusha Reddy Thumma- (pthumma)
2. Kavya Sri Kasarla -(kkasarla)
3. Bharath chowdary Anumolu-(banumolu)[1]*

Abstract
BACKGROUND: Face recognition, handwritten digit recognition has become an appreciable area of research for the practical approaches but this task of making new approaches has become difficult due to the existing data problem and defect in the the data.So, this classification of faces and digits make automatic classification a difficult tasks. Classify task with higher accuracy and proposing models for dynamism data sets has become a challenging task.
OBJECTIVE: Deep Convolutional Neural Networks (DCNNs) have emerged as a popular architecture for facial emotion classification due to their ability to learn complex features directly from raw image data. DCNNs have demonstrated state-of-the-art performance on various facial emotion datasets, including the FER2013 dataset, which contains over 35,000 facial expression images.
METHODS: In this project, we aim to develop a DCNN model for the classification of facial emotions using the dataset. We employ various techniques like ELU activation, he-normal kernel initializer, dropouts, and batch normalization to improve the model's generalization performance.
The exploration of data augmentation techniques to further enhance the model's performance. The developed model can potentially aid in emotion recognition and effective computing applications, contributing towards a better understanding of human-computer interaction and the digit classification aims to print the digit identification towards stored database.

Keywords
Convolution layer, Deep Convolution Neural Network, FER2013 data set, ELU activation

[1] *Luddy School of Informatics, Computing, and Engineering, Indiana University, Bloomington, IN, USA*

Contents

1. Problem and Data Description

Face Digit recognition and classification play sin important role in today's life. It has become the medium of communication to connect with people and to present someone's idea in the form of text. So, identification has become a necessary in all kinds of areas. The diversity often makes it difficult to recognize and read/identify. The necessity of recognition of images and digits is necessary and thus several deep learning algorithms were implemented. They provide better results than conventional methods and these algorithms require less time to recognize especially when we have large data-sets. For this project we have taken MNIST data-set for digit classification and FER2013 data set for image classification , the reason that it is chosen is that, it has all the data pre-processed and when we apply the models then we could trigger the exact output for classification.

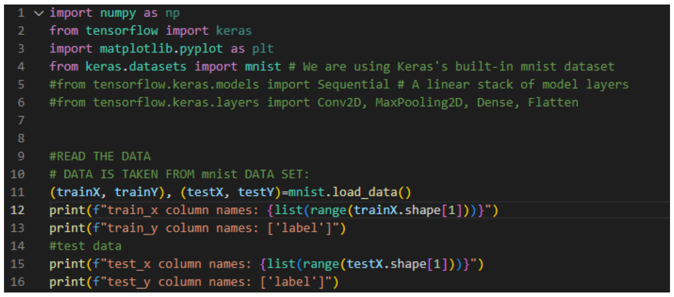
DATA DESCRIPTION:
FACE EMOTION CLASSIFICATION:

• The dataset contains grayscale images of faces, each with a size of 48x48 pixels.
• The faces in the images are centered and occupy similar amounts of space.
• The goal is to classify each image into one of seven emotion categories: Angry, Disgust, Fear, Happy, Sad, Surprise, and Neutral.
• The dataset has two files: train.csv and test.csv.
• The train.csv file contains two columns: "emotion" and "pixels".
• The "emotion" column has a number between 0 and 6 indicating the emotion in the image.
• The "pixels" column has a string with space-separated pixel

values in row-major order.
• The test.csv file has only the ”pixels” column, and the task is to predict the emotion for each image.
• The training set has 28,709 examples, while the public test set has 3,589 examples.
DIGIT CLASSIFICATION:
• MNIST data: 70000 garyscale images of handwritten digits (0-9) Training data-set: 60000 images
Testing data-set: 10000 images
Image scaling: 28x28- pixel square represented as a 2D array of grayscale pixel range: 0-255
Standardization: pixel values are normalized
Mean= 0.1307
Standard deviation= 0.3081

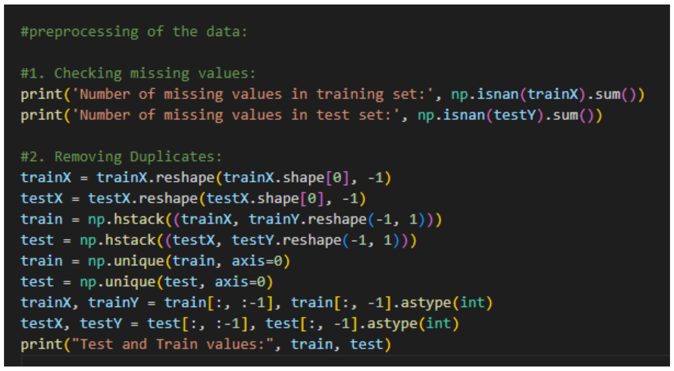
2. Data Preprocessing & Exploratory Data Analysis

The following steps are followed on MNSIT data-set for Data Pre-processing:
1. Load the data:
In this code, we use the **mnist.loaddata()** function from the Keras.datasetsmoduletoloadtheMNIST dataset.



```python
import numpy as np
from tensorflow import keras
import matplotlib.pyplot as plt
from keras.datasets import mnist # We are using Keras's built-in mnist dataset
#from tensorflow.keras.models import Sequential # A linear stack of model layers
#from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten


#READ THE DATA
# DATA IS TAKEN FROM mnist DATA SET:
(trainX, trainY), (testX, testY)=mnist.load_data()
print(f"train_x column names: {list(range(trainX.shape[1]))}")
print(f"train_y column names: ['label']")
#test data
print(f"test_x column names: {list(range(testX.shape[1]))}")
print(f"test_y column names: ['label']")
```

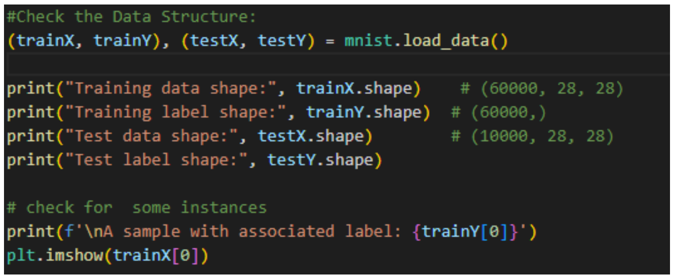
2.Data cleaning:
MNIST dataset is a well-curated dataset that has already undergone some cleaning and preprocessing, there are still some common data cleaning steps that can be taken to improve the quality of the data. Some of these steps include:
1. Checking missing values:
Check for any missing or corrupted data in the dataset and either remove the corresponding examples or impute the missing values with appropriate replacements. The following steps are followed on face image data-set for Data Pre-processing:



```python
#preprocessing of the data:

#1. Checking missing values:
print('Number of missing values in training set:', np.isnan(trainX).sum())
print('Number of missing values in test set:', np.isnan(testY).sum())

#2. Removing Duplicates:
trainX = trainX.reshape(trainX.shape[0], -1)
testX = testX.reshape(testX.shape[0], -1)
train = np.hstack((trainX, trainY.reshape(-1, 1)))
test = np.hstack((testX, testY.reshape(-1, 1)))
train = np.unique(train, axis=0)
test = np.unique(test, axis=0)
trainX, trainY = train[:, :-1], train[:, -1].astype(int)
testX, testY = test[:, :-1], test[:, -1].astype(int)
print("Test and Train values:", train, test)
```

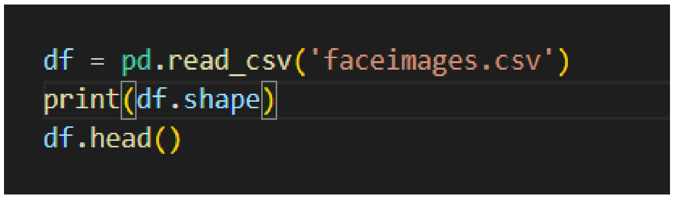
2. Removing duplicates:
Check for any duplicate entries in the data-set and remove them to ensure that each image and label is unique.
3. Remove outliers
4. Checking for data consistency
5. Standardization

3. Check the Data Structure:
The MNIST dataset can be loaded into Python using various libraries such as tensorflow, pytorch, or sklearn.



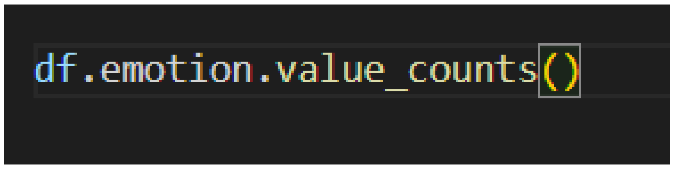
```python
#Check the Data Structure:
(trainX, trainY), (testX, testY) = mnist.load_data()

print("Training data shape:", trainX.shape)    # (60000, 28, 28)
print("Training label shape:", trainY.shape)  # (60000,)
print("Test data shape:", testX.shape)        # (10000, 28, 28)
print("Test label shape:", testY.shape)

# check for  some instances
print(f'\nA sample with associated label: {trainY[0]}')
plt.imshow(trainX[0])
```

Consider the pre proceessing for Face Classification:
1. Load the data: In this code, we use **pd.read-csv** function for faceimages.csv data set.



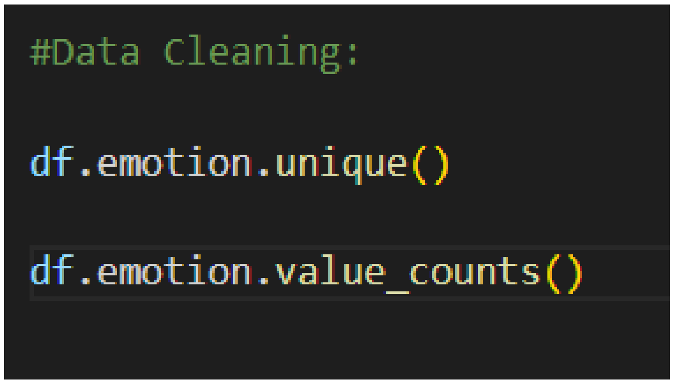
```python
df = pd.read_csv('faceimages.csv')
print(df.shape)
df.head()
```

2.Data cleaning: We ahve taken th edata from kaggle and used some cleaning and pre-processing, data cleaning steps that can be taken to improve the quality of the data. Some of these steps include:
1. Checking missing values: Check for any missing or corrupted data in the data set and either remove the corresponding examples or impute the missing values with appropriate replacements.



```
df.emotion.value_counts()
```

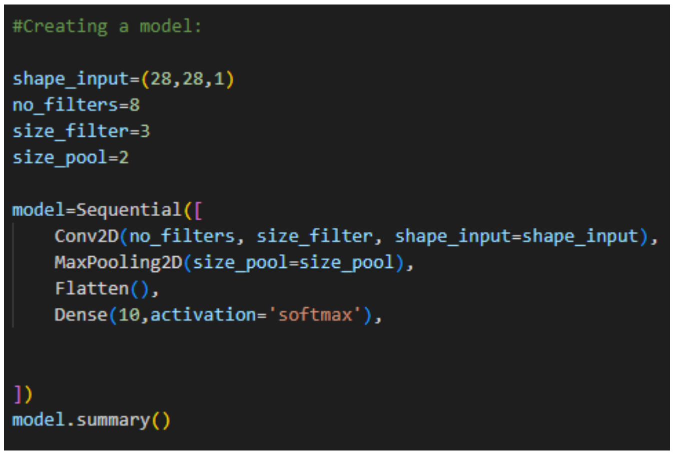
2. Removing duplicates:
Check for any duplicate entries in the data-set and remove them to ensure that each image and label is unique.



```
#Data Cleaning:

df.emotion.unique()

df.emotion.value_counts()
```

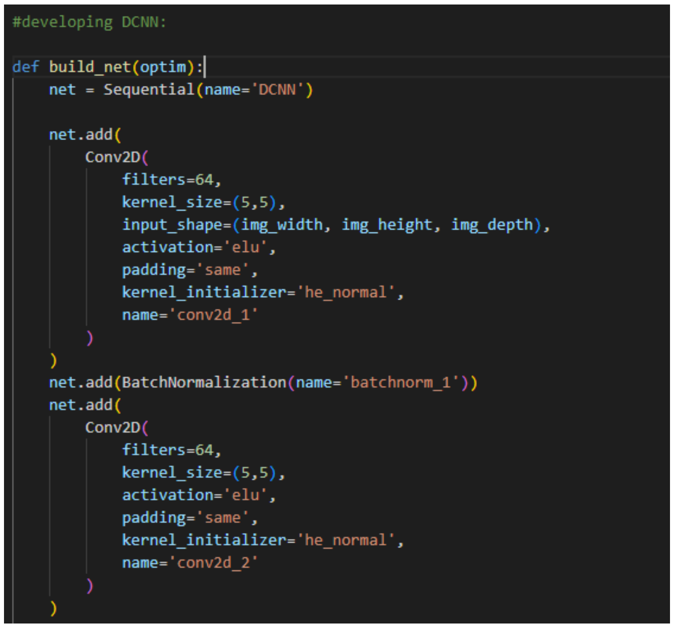
3. Algorithm and Methodology

ALGORITHM - MODEL CREATION FOR DIGIT CLASSIFICATION:
Creating a Model:
Creating a model on the MNIST data set involves building a machine learning or deep learning model that can accurately classify the handwritten digits in the data set. We use simple Convolution Neural Network (CNN) using " tensorflow" to classify the data.
The CNN architecture consists of the following layers:
1.Conv2D:
This layer applies convolution to the input image using the specified number of filters and filter size.
2.MaxPooling2D:
This layer performs max pooling operation over a 2x2 window to reduce the spatial dimensions of the output from the convolution layer.
3.Flatten:
This layer flattens the output from the previous layer, converting it into a one-dimensional vector.
4.Dense:
This layer applies a fully connected neural network with 10 output units and softmax activation function. The inputShape variable defines the shape of the input image as 28x28 with one channel (grayscale). The numberOfFilters variable specifies the depth of the convolutional layer, which is set to 8. The filterSize variable specifies the size of the filter, which is set to 3x3. The poolSize variable specifies the size of the max pooling window, which is set to 2x2. The model.summary() function prints a summary of the CNN architecture, including the shape of the output from each layer and the total number of parameters in the model.



```
#Creating a model:

shape_input=(28,28,1)
no_filters=8
size_filter=3
size_pool=2

model=Sequential([
    Conv2D(no_filters, size_filter, shape_input=shape_input),
    MaxPooling2D(size_pool=size_pool),
    Flatten(),
    Dense(10,activation='softmax'),


])
model.summary()
```

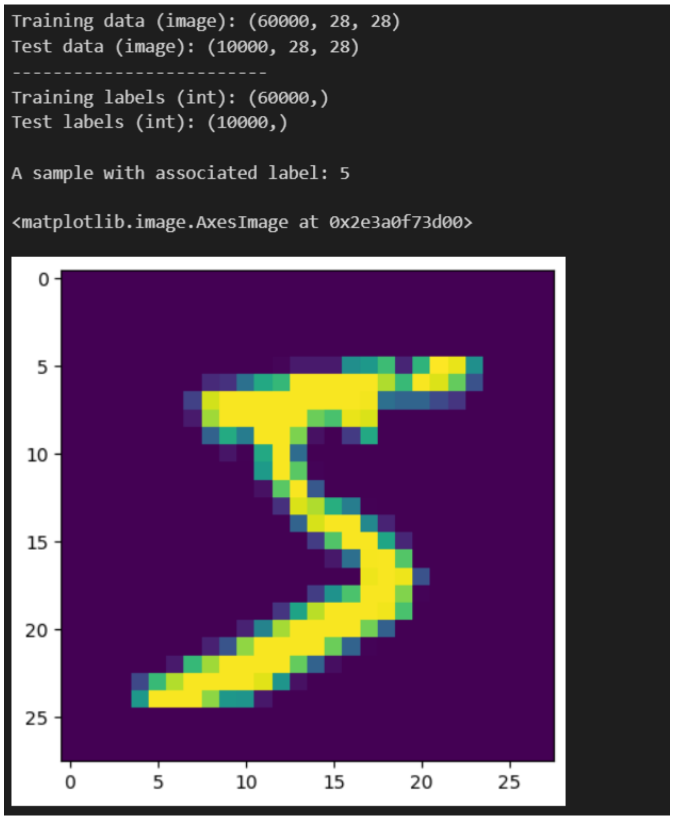
Creating a Model: Creating a model on the given data set involves building a machine learning or deep learning model that can accurately classify the images in the data set.
This is a description of a Deep Convolutional Neural Network (DCNN) used in a particular case. The network includes dropout layers at regular intervals to aid generalization. The activation function used is ELU, which avoids the dying ReLU problem and has been found to perform well in this case compared to LeakyReLU. The henormal kernel initializer is used, as it is suitable for use with ELU. Batch normalization is also used to improve the results.



```
#developing DCNN:

def build_net(optim):
    net = Sequential(name='DCNN')

    net.add(
        Conv2D(
            filters=64,
            kernel_size=(5,5),
            input_shape=(img_width, img_height, img_depth),
            activation='elu',
            padding='same',
            kernel_initializer='he_normal',
            name='conv2d_1'
        )
    )
    net.add(BatchNormalization(name='batchnorm_1'))
    net.add(
        Conv2D(
            filters=64,
            kernel_size=(5,5),
            activation='elu',
            padding='same',
            kernel_initializer='he_normal',
            name='conv2d_2'
        )
    )
```

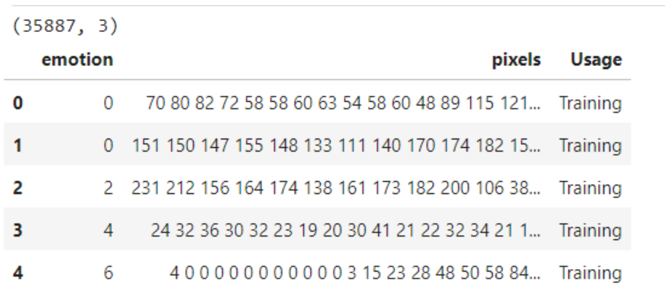
4. Experiments and Results

1. Data Structure:
For every data-set we train we need to check the data structure so as to fit all the attributes without missing values:
2. Model implementation:
The shape-input variable is a tuple that specifies the shape of the input images. In this case, the input images are 28x28 grayscale images, so shape-input is set to (28, 28, 1). The no-filters variable specifies the number of filters to use in the convolutional layer. In this case, we use 8 filters. The size-filter variable specifies the size of the filters. In this case, the filter size is 3x3. The size-pool variable specifies the size of the max pooling window. In this case, the max pooling window is 2x2. The model object is created using the Sequential class from Keras. This class allows us to define a linear stack of layers. The first layer in the model is a Conv2D layer. This layer applies convolutional filters to the input images. We specify the number of filters and the filter size using the no-filters and size-filter variables, and we pass in the shape-input variable to specify the shape of the input images. The second layer in the model is a MaxPooling2D layer. This layer performs max pooling on the output from the convolutional layer. We specify the size of the max pooling window using the size-pool variable. The third layer in the model is a Flatten layer. This layer flattens the output from the previous layer into a one-dimensional vector. The fourth and final layer in the model is a Dense layer with 10 output units and a soft max activation function. This layer applies a fully connected neural network to the flattened output from the previous layer, and the softmax activation function produces a probability distribution over the 10 possible classes. The model.summary() function prints a summary of the model Semester Project — 4/4 architecture, including the shape of the output from each layer and the total number of parameters in the model. This can be useful for debugging and optimizing the model.



```
Training data (image): (60000, 28, 28)
Test data (image): (10000, 28, 28)
------------------------
Training labels (int): (60000,)
Test labels (int): (10000,)

A sample with associated label: 5

<matplotlib.image.AxesImage at 0x2e3a0f73d00>
```

Loading the data:

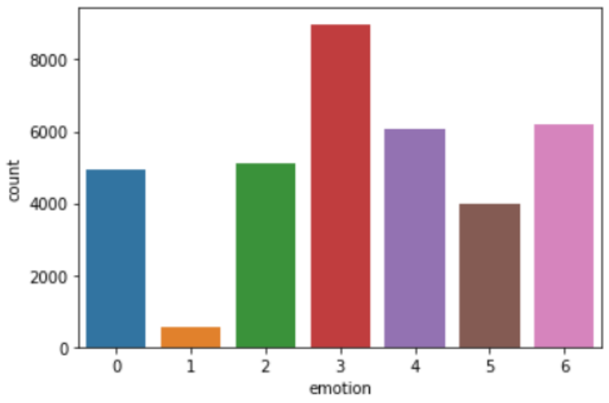


```
(35887, 3)
```

	emotion	pixels	Usage
0	0	70 80 82 72 58 58 60 63 54 58 60 48 89 115 121...	Training
1	0	151 150 147 155 148 133 111 140 170 174 182 15...	Training
2	2	231 212 156 164 174 138 161 173 182 200 106 38...	Training
3	4	24 32 36 30 32 23 19 20 30 41 21 22 32 34 21 1...	Training
4	6	4 0 0 0 0 0 0 0 0 0 0 0 3 15 23 28 48 50 58 84...	Training

Creating labels:
The sns.countplot() function plots the count of each unique value in the specified column, in this case, the emotion labels. The resulting plot shows the number of images in the dataset that correspond to each emotion.

The pyplot.show() function displays the resulting plot. This code is useful to get an idea of the distribution of the different emotions in the dataset and can be used to check if the dataset is balanced or imbalanced with respect to the different classes.

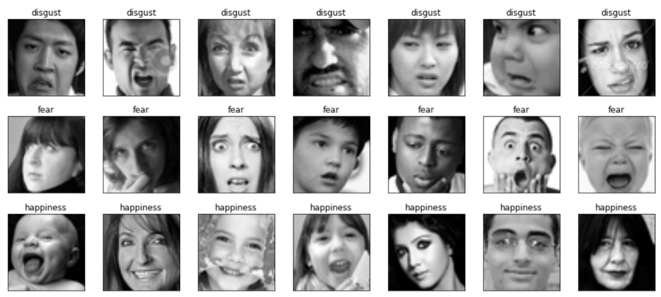


Grid creation-emotion labels:
The fig = pyplot.figure(1, (14, 14)) line creates a figure object with a size of 14x14 inches, which will be used to display the grid of images.

The nested for-loops iterate over each unique emotion label in the dataset and display an image for each label in its corresponding row in the grid.

The ax.imshow() function displays the image in grayscale using the cmap='gray' parameter. The ax.set-xticks([]) and ax.set-yticks([]) lines remove the tick marks on the x and y axes. The ax.set-title() function sets the title of each cell in the grid to the corresponding emotion label.

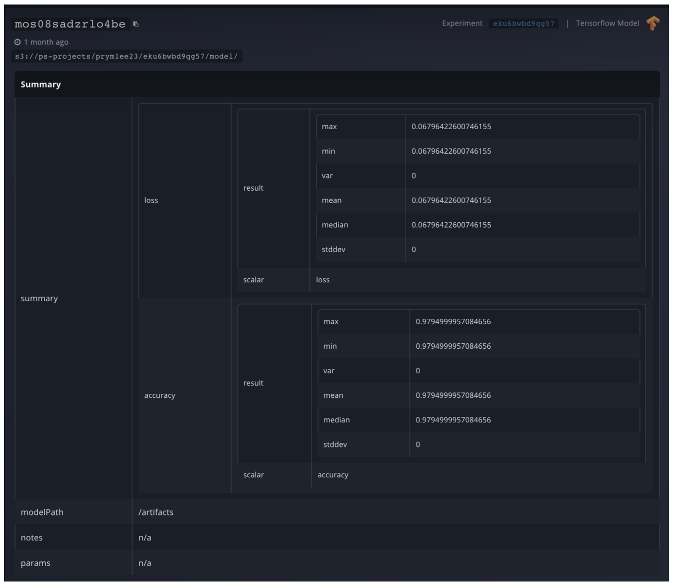
The pyplot.tight-layout() function adjusts the spacing between subplots to improve the overall layout of the grid of images. This code is useful to visualize a sample of the images in the dataset and get an idea of what the images look like for each emotion class.



5. Deployment and Maintenance

We used **https://docs.paperspace.com/gradient/models/** and we are working on the deployment on free platform
Face Classification:
Face classification involves identifying individuals by their facial features. This can be done using computer vision techniques and machine learning algorithms. The following are some common steps in deploying a face classification model:

Data Collection:
Collecting a large dataset of face images is the first step. This dataset will be used to train and validate the face classification model.
Data Preprocessing:
Preprocessing techniques such as normalization, cropping, and resizing are applied to the dataset to ensure consistency and improve the performance of the model.
Model Training:
Machine learning algorithms such as convolutional neural networks (CNN) are used to train the model on the preprocessed dataset. Model Testing and Validation: The model is tested and validated using a separate set of images that were not used in the training process.
Deployment:
The model can be deployed on a web application, mobile application or as part of a larger system.
Digit Classification:
Digit classification involves identifying handwritten digits from an image. This can be done using machine learning algorithms. The following are some common steps in deploying a digit classification model:
Data Collection:
Collecting a large dataset of handwritten digit images is the first step. This dataset will be used to train and validate the digit classification model. **Data Preprocessing:**
Preprocessing techniques such as normalization, binarization, and resizing are applied to the dataset to ensure consistency and improve the performance of the model.
Model Training:
Machine learning algorithms such as support vector machines (SVM) or decision trees are used to train the model on the preprocessed dataset.
Model Testing and Validation:
The model is tested and validated using a separate set of images that were not used in the training process.
Deployment:
The model can be deployed on a web application, mobile application or as part of a larger system.


mos08sadzrlo4be

1 month ago

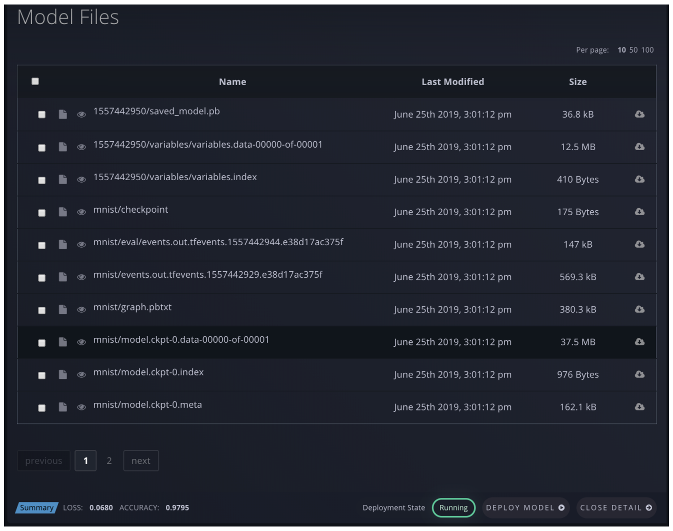
s3://ps-projects/prymlee23/eku6bwbd9qg57/model/

Experiment eku6bwbd9qg57 | Tensorflow Model

Summary

summary	loss	result	max	0.06796422600746155
			min	0.06796422600746155
			var	0
			mean	0.06796422600746155
			median	0.06796422600746155
			stddev	0
		scalar	loss	
	accuracy	result	max	0.9794999957084656
			min	0.9794999957084656
			var	0
			mean	0.9794999957084656
			median	0.9794999957084656
			stddev	0
		scalar	accuracy	
modelPath	/artifacts			
notes	n/a			
params	n/a			

In both cases, the accuracy of the model can be improved by using a larger and more diverse dataset, fine-tuning the model hyperparameters, and incorporating new techniques such as transfer learning.


Model Files

Per page: 10 50 100

Name	Last Modified	Size
1557442950/saved_model.pb	June 25th 2019, 3:01:12 pm	36.8 kB
1557442950/variables/variables.data-00000-of-00001	June 25th 2019, 3:01:12 pm	12.5 MB
1557442950/variables/variables.index	June 25th 2019, 3:01:12 pm	410 Bytes
mnist/checkpoint	June 25th 2019, 3:01:12 pm	175 Bytes
mnist/eval/events.out.tfevents.1557442944.e38d17ac375f	June 25th 2019, 3:01:12 pm	147 kB
mnist/events.out.tfevents.1557442929.e38d17ac375f	June 25th 2019, 3:01:12 pm	569.3 kB
mnist/graph.pbtxt	June 25th 2019, 3:01:12 pm	380.3 kB
mnist/model.ckpt-0.data-00000-of-00001	June 25th 2019, 3:01:12 pm	37.5 MB
mnist/model.ckpt-0.index	June 25th 2019, 3:01:12 pm	976 Bytes
mnist/model.ckpt-0.meta	June 25th 2019, 3:01:12 pm	162.1 kB

6. Summary and Conclusions

Face classification models can be used for a variety of applications, such as security systems, identity verification, and personalized marketing. The performance of the face classification model heavily depends on the quality and diversity of the training dataset. As with any machine learning model, it is important to ensure that the face classification model is fair, transparent, and secure. Digit classification models can be used for a variety of applications, such as recognizing postal codes, sorting mail, and recognizing bank checks. The performance of the digit classification model heavily depends on the quality and diversity of the training dataset. As with any machine learning model, it is important to ensure that the digit classification model is fair, transparent, and secure.

Acknowledgments

We would like to thank Professor and all the Teaching Assistants who gave us idea on how to proceed with the outline of the project and detailing on working sources.

References

1.https://www.researchgate.net/publication/
2.https://arxiv.org/ftp/arxiv/papers/1505/1505.04058.pdf