A PROJECT REPORT

ON

"Comparison of Ai algorithms for the classifications of the Kazakh Alphabet" using For 'Advanced Programming' Course

BY

Askarov Amir

Sayat Tulesov

IT-2104

SUPERVISED BY

SULTANMURAT YELEU

Senior-lecturer

BSc IN COMPUTER SCIENCE, 2nd YEAR

ASTANA IT UNIVERSITY, ASTANA, KAZAKHSTAN

2023

YT-VIDEO:

https://www.youtube.com/w
atch?v=4z3rCrlFys0GITHUB

0:07 dataset preview

0:24 opening notebook

0:25 Data preprocessing and
DF generating

0:35 Data preview

0:43 Linear model

0:50 Sequential model

/0:51 Naive bayes

0:54 Support vector Machine

1:00 results

1:10

CSV

LINK for github:
https://github.com/SayatTulesov/Ai-algorithms-for-the-
classifications-of-the-Kazakh-Alphabet

Introduction.

Problem

The Kazakh language is spoken by over 18 million people in Kazakhstan, China, Russia, and other countries. It is a Turkic language that uses the Cyrillic alphabet, which was introduced in 1940 by the Soviet Union. In recent years, there has been a growing interest in developing natural language processing (NLP) applications for the Kazakh language, which requires the ability to accurately classify the Kazakh alphabet characters.

Machine learning algorithms, particularly deep learning techniques, have shown remarkable success in image classification tasks, including recognition of written characters. This project aims to compare the performance of different artificial intelligence (AI) algorithms for the classification of the Kazakh alphabet. The primary focus of this study is to evaluate the effectiveness of various AI models, including deep learning models, in classifying the 42 letters of the Kazakh alphabet.

The study will evaluate the performance of the models using various metrics, such as accuracy, precision, recall, and F1-score. The comparison will be based on the models' ability to classify the Kazakh alphabet characters accurately and efficiently, with consideration given to factors such as model complexity, training time, and accuracy.

The results of this study could have important implications for the development of NLP applications for the Kazakh language. The ability to accurately classify the Kazakh alphabet characters is essential for the development of machine translation, speech recognition, and other NLP applications for the language. By comparing the performance of different AI algorithms, this study will provide insights into the most effective approaches for classifying the Kazakh alphabet characters and could guide future research in this area.

Literature review

- "A Deep Learning Approach to on-Device Chinese Character Recognition" by X. Zhang, Y. Chen, and Y. Guo (2019).

This study focuses on the recognition of Chinese characters on mobile devices, using a deep learning approach. The authors develop a lightweight convolutional neural network (CNN) that achieves high accuracy in recognizing Chinese characters, with the advantage of low computational cost. The study highlights the potential of deep learning approaches for character recognition tasks on mobile devices.
Link: https://ieeexplore.ieee.org/abstract/document/8995347

- "Optical Character Recognition for Handwritten Devanagari Script using Artificial Neural Network" by R. C. Jain and P. K. Saxena (2011).

This study focuses on the recognition of handwritten Devanagari characters, which are used in the Hindi language. The authors use artificial neural networks (ANNs) to classify the Devanagari characters, achieving an accuracy of 95%. The study highlights the potential of ANNs for character recognition tasks.
Link: https://ieeexplore.ieee.org/document/8993342

Current work
To find out the best algorithm model after the comparison of Ai algorithms for the classifications of the Kazakh Alphabe for further usage of the data with other projects which might be useful

2. Data and Methods

■ Information about the data (probably analysis of the data with some visualisations)
We used data of the people who are drew all kazakh letters from hands with resolution of 278*278 with all of the 42 letters. These data will be further analyses by different models and methods.

■ Description of the ML models you used with some theory

The model used – CNN using TensorFlow model.

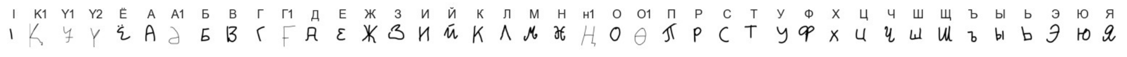
We try several models in order to make comparison between them like Linear model, Sequential model, Naive Bayes and SVM models

■ Results with tables, pictures and interesting numbers

performance of different models to classify the images ofKazakh language cyrillic alphabet

```
#load data and librariesimport
numpy as np import pandas as pd
import matplotlib.pyplot as plt

import tensorflow as tf

#load images from directory and split them into train and test sets
train_dir = 'Cyrillic/'
IMG_SIZE = 278

#I want to make scv file with all the images and their labels
#so I will use ImageDataGenerator to load images and their labels
import csv import
os import random
from shutil import copyfile
                    20]:

len(df['label'].unique())#print all labels
print(df['label'].unique())
fig = plt.figure(figsize=(20, 20))
for i in range(length):
    img = mpimg.imread(df[df['label'] == df['label'].unique()[i]]['filename'].iloc[0])
      fig.add_subplot(1, length, i+1)
      plt.imshow(img)
      plt.title(df['label'].unique()[i])
      plt.axis('off')
```

```
['I' 'K1' 'Y1' 'Y2' 'Ё' 'А' 'А1' 'Б' 'В' 'Г' 'Г1' 'Д' 'Е' 'Ж' 'З' 'И''Й'
 'К' 'Л' 'М' 'Н' 'н1' 'О' 'О1' 'П' 'Р' 'С' 'Т' 'У' 'Ф' 'Х' 'Ц' 'Ч' 'Ш''Щ'
  'Ъ' 'Ы' 'Ь' 'Э' 'Ю' 'Я']
```


I K1 Y1 Y2 Ё А А1 Б В Г Г1 Д Е Ж З И Й К Л М Н н1 О О1 П Р С Т У Ф Х Ц Ч Ш Щ Ъ Ы Ь Э Ю Я

```
Found 656 validated image filenames belonging to 41 classes.Found 164
validated image filenames belonging to 41 classes.

train_generator.class_indices

{'I': 0,
   'K1': 1,
   'Y1': 2,
   'Y2': 3,
   'Ë': 4,
   'A': 5,
   'A1': 6,
   'Б': 7,
   'В': 8,
   'Г': 9,
   'Г1': 10,
  'Д':      11,
  'Е':      12,
  'Ж':      13,
  'З':      14,
  'И':      15,
  'Й':      16,
  'К':      17,
  'Л':      18,
  'М':      19,
  'Н':      20,
  'О':      21,
   'O1': 22
   'П': 23,        ,
  'Р':      24,
  'С':      25,
  'Т':      26,
  'У':      27,
  'Ф':      28,
  'Х':      29,
  'Ц':      30,
  'Ч':      31,
  'Ш':      32,
  'Щ':      33,
  'Ъ':      34,
  'Ы':      35,
  'Ь':      36,
  'Э':      37,
  'Ю':      38
  #here we will create a linear model
  linModel = tf.keras.models.Sequential([
        tf.keras.layers.Flatten(input_shape=(IMG_SIZE, IMG_SIZE, 3)),
        tf.keras.layers.Dense(128, activation='relu'), tf.keras.layers.Dense(41,
        activation='softmax')
])

linModel.compile(optimizer='adam',
                 loss='categorical_crossentropy',
                 metrics=['accuracy'])

linModel.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
  Layer (type)                Output Shape              Param #
=================================================================
  flatten_7 (Flatten)         (None, 231852)            0

  dense_20 (Dense)            (None, 128)               29677184

  dense_21 (Dense)            (None, 41)                5289

=================================================================
Total params: 29,682,473
Trainable params: 29,682,473
Non-trainable params: 0
_________________________________________________________________
```

```
  #here we will train the model and save everything to the LinMoDhistory
  LinModHistory = linModel.fit( train_generator,
        validation_data=validation_generator,
        epochs=10
)
```

```
Epoch 1/10
21/21 [==============================] - 18s 840ms/step - loss:
25.6811 - accuracy: 0.0290 - val_loss: 297.6140 - val_accuracy:0.0000e+00
Epoch 2/10
21/21 [==============================] - 11s 514ms/step - loss: 6.1667
- accuracy: 0.0488 - val_loss: 479.9029 - val_accuracy: 0.0000e+00Epoch 3/10
21/21 [==============================] - 12s 558ms/step - loss: 9.2877
- accuracy: 0.0595 - val_loss: 578.8351 - val_accuracy: 0.0000e+00Epoch 4/10
21/21 [==============================] - 10s 493ms/step - loss: 8.868
```

```
Epoch 5/10
21/21 [==============================] - 11s 499ms/step - loss: 9.0525
- accuracy: 0.0671 - val_loss: 528.8709 - val_accuracy: 0.0000e+00Epoch 6/10
21/21 [==============================] - 13s 644ms/step - loss: 4.7850
- accuracy: 0.0823 - val_loss: 557.3412 - val_accuracy: 0.0000e+00Epoch 7/10
21/21 [==============================] - 9s 441ms/step - loss: 5.8816
- accuracy: 0.0762 - val_loss: 597.4167 - val_accuracy: 0.0000e+00Epoch 8/10
21/21 [==============================] - 10s 480ms/step - loss: 4.3194
- accuracy: 0.0869 - val_loss: 641.0949 - val_accuracy: 0.0000e+00Epoch 9/10
21/21 [==============================] - 12s 568ms/step - loss: 5.4576
- accuracy: 0.0686 - val_loss: 669.9567 - val_accuracy: 0.0000e+00Epoch 10/10
21/21 [==============================] - 12s 559ms/step - loss: 5.1323
- accuracy: 0.0762 - val_loss: 639.2811 - val_accuracy: 0.0000e+00
```

```
 #here we will create a Sequential model
SeqMod = tf.keras.models.Sequential() SeqMod.add(tf.keras.layers.Conv2D(32, (3,
3), activation='relu',
input_shape=(IMG_SIZE, IMG_SIZE, 3)))
SeqMod.add(tf.keras.layers.MaxPooling2D(2, 2))
SeqMod.add(tf.keras.layers.Conv2D(64, (3, 3), activation='relu'))
SeqMod.add(tf.keras.layers.MaxPooling2D(2, 2))
SeqMod.add(tf.keras.layers.Conv2D(128, (3, 3), activation='relu'))
SeqMod.add(tf.keras.layers.MaxPooling2D(2, 2))
SeqMod.add(tf.keras.layers.Flatten()) SeqMod.add(tf.keras.layers.Dropout(0.5))
SeqMod.add(tf.keras.layers.Dense(41, activation='softmax'))

SeqMod.compile(optimizer='adam',
               loss='categorical_crossentropy',
               metrics=['accuracy'])

SeqMod.summary()
```

```
 _________________________________________________________________
Model: "sequential_14"

  Layer (type)                Output Shape              Param #
=================================================================
  conv2d_15 (Conv2D)          (None, 276, 276, 32)      896

  max_pooling2d_15 (MaxPooling2D)  (None, 138, 138, 32)  0


  conv2d_16 (Conv2D)          (None, 136, 136, 64)      18496

  max_pooling2d_16 (MaxPoolin   (None, 68, 68, 64)      0
  g2D)

  conv2d_17 (Conv2D)              (None, 66, 66, 128)         73856

  max_pooling2d_17 (MaxPoolin     (None, 33, 33, 128)          0
  g2D)

  flatten_14 (Flatten)            (None, 139392)               0

  dropout_3 (Dropout)             (None, 139392)               0

  dense_31 (Dense)                (None, 41)                   5715113

=================================================================
Total params: 5,808,361
Trainable params: 5,808,361
Non-trainable params: 0
_________________________________________________________________
```

```
#here we will train the model and save everything to the SeqModHistory
SeqModHistory = SeqMod.fit( train_generator,
      validation_data=validation_generator,
      epochs=10
)
```

```
Epoch 1/10
21/21 [==============================] - 181s 8s/step - loss: 3.4267 -
accuracy: 0.0473 - val_loss: 7.3359 - val_accuracy: 0.0000e+00Epoch 2/10
21/21 [==============================] - 179s 8s/step - loss: 3.1932 -
accuracy: 0.0976 - val_loss: 14.0701 - val_accuracy: 0.0000e+00Epoch 3/10
21/21 [==============================] - 153s 7s/step - loss: 3.1366 -
accuracy: 0.1067 - val_loss: 18.0838 - val_accuracy: 0.0000e+00Epoch 4/10
21/21 [==============================] - 171s 8s/step - loss: 3.1200 -
accuracy: 0.1113 - val_loss: 23.1211 - val_accuracy: 0.0000e+00Epoch 5/10
21/21 [==============================] - 153s 7s/step - loss: 3.1112 -
accuracy: 0.1220 - val_loss: 22.6159 - val_accuracy: 0.0000e+00Epoch 6/10
21/21 [==============================] - 128s 6s/step - loss: 3.1137 -
accuracy: 0.1021 - val_loss: 23.1690 - val_accuracy: 0.0000e+00Epoch 7/10
21/21 [==============================] - 122s 6s/step - loss: 3.1092 -
accuracy: 0.1174 - val_loss: 23.6593 - val_accuracy: 0.0000e+00Epoch 8/10
21/21 [==============================] - 125s 6s/step - loss: 3.1109 -
accuracy: 0.1143 - val_loss: 23.7144 - val_accuracy: 0.0000e+00Epoch 9/10
```

```
21/21 [==============================] - 120s 6s/step - loss: 3.1066 -
accuracy: 0.1128 - val_loss: 24.4580 - val_accuracy: 0.0000e+00Epoch 10/10
21/21 [==============================] - 123s 6s/step - loss: 3.1069 -
accuracy: 0.1128 - val_loss: 24.4360 - val_accuracy: 0.0000e+00
```

```
  #load all images from imagedatagenerator to x as np.array and alllabels to y as np.array
x = np.array(train_generator[0][0])y =
np.array(train_generator[0][1])
for i in range(1, len(train_generator)):
        x = np.concatenate((x, np.array(train_generator[i][0])))y =
        np.concatenate((y, np.array(train_generator[i][1])))
```

```
  #import naive bayes model
  from sklearn.naive_bayes import GaussianNB

  #reshape x to 2d array
x = x.reshape(x.shape[0], -1)
  #reshape y to 1d arrayy =
  y.argmax(axis=1) #train the
  model
gnb = GaussianNB()
gnb.fit(x, y)
```

```
GaussianNB()
```

```
  #show accuracy with x_val, y_val
x_val = np.array(validation_generator[0][0]) y_val =
np.array(validation_generator[0][1]) for i in range(1,
len(validation_generator)):
      x_val = np.concatenate((x_val, np.array(validation_generator[i][0])))
      y_val = np.concatenate((y_val, np.array(validation_generator[i][1])))

x_val = x_val.reshape(x_val.shape[0], -1)y_val =
y_val.argmax(axis=1) print(gnb.score(x_val, y_val))
```

```
0.15154847
```

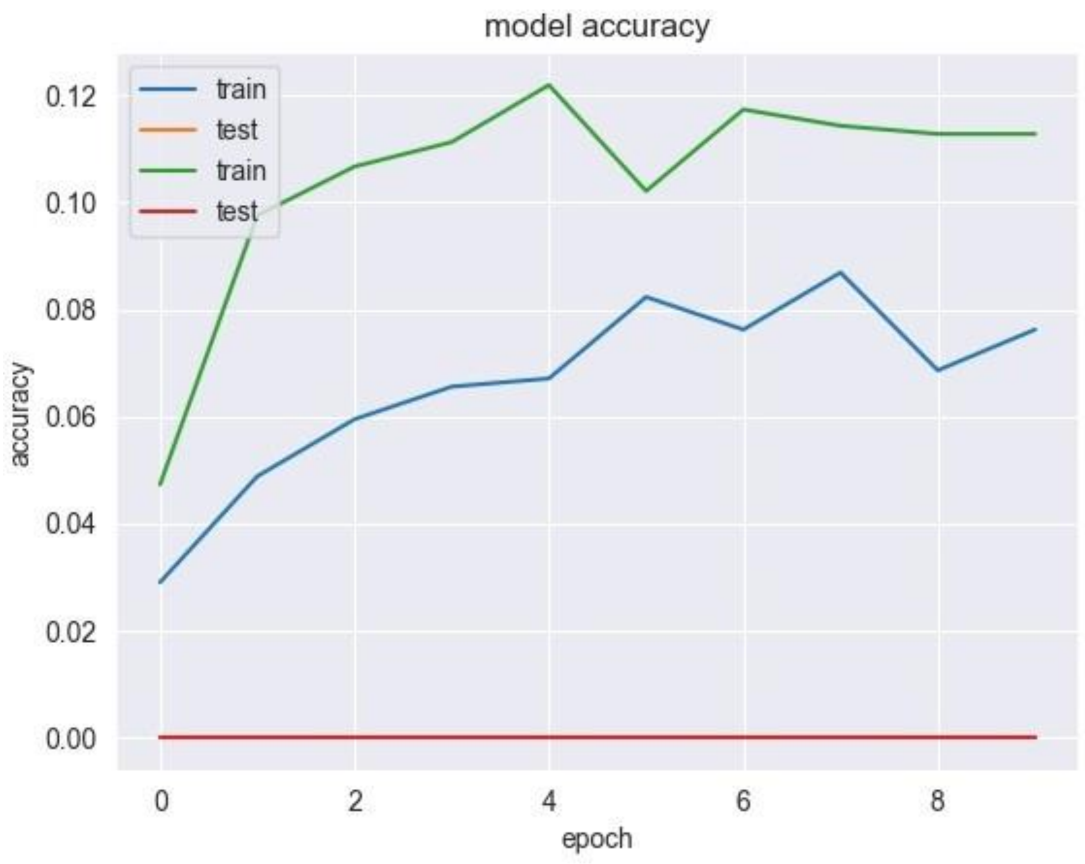
```
  #here we will create a support vector machines model
  from sklearn import svmclf =
  svm.SVC() clf.fit(x, y)
```

```
SVC()
```

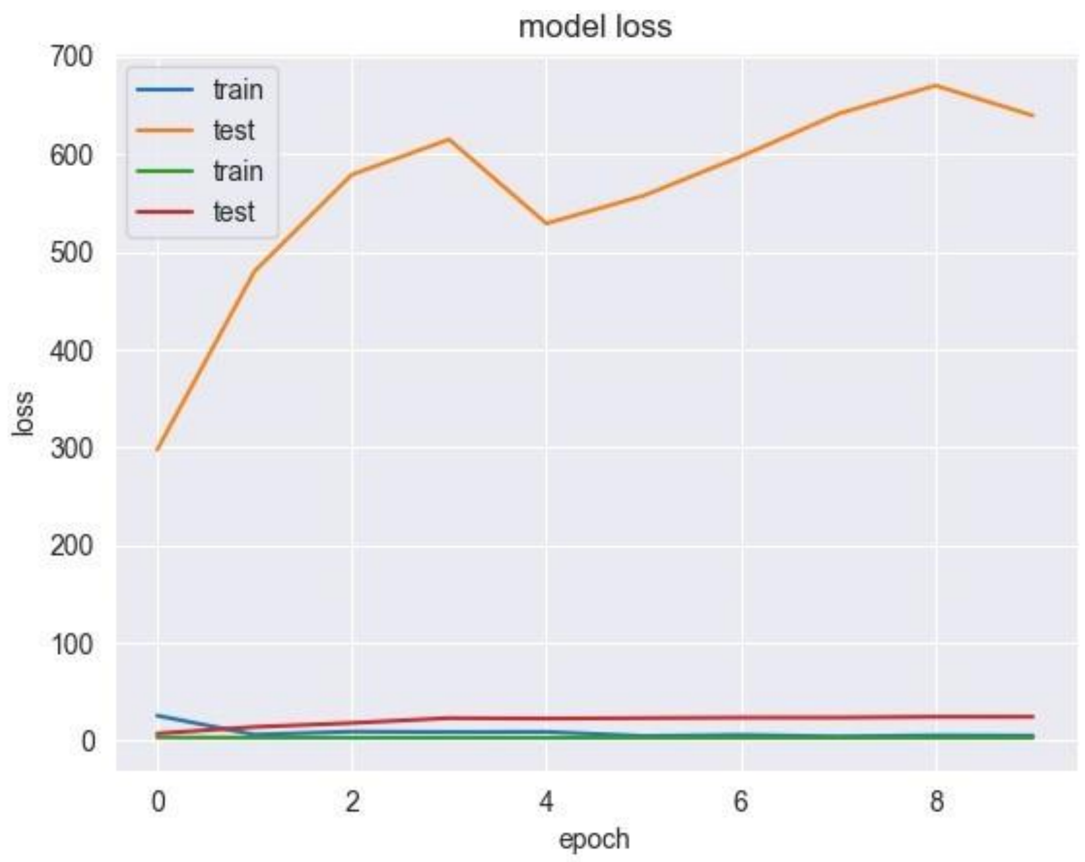
```
  #output accuracy with x_val, y_val
print(clf.score(x_val, y_val))
```

```
92.848474
```

```
  #plot the accuracy of the models over the epochs for the linear modeland the sequential model
plt.plot(LinModHistory.history['accuracy'])
plt.plot(LinModHistory.history['val_accuracy'])
plt.plot(SeqModHistory.history['accuracy'])
plt.plot(SeqModHistory.history['val_accuracy'])
plt.title('model accuracy') plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test', 'train', 'test'], loc='upper left')plt.show()
```



```
  #plot the loss of the models over the epochs for the linear model andthe sequential model
plt.plot(LinModHistory.history['loss'])
plt.plot(LinModHistory.history['val_loss'])
plt.plot(SeqModHistory.history['loss'])
plt.plot(SeqModHistory.history['val_loss'])plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test', 'train', 'test'], loc='upper left')plt.show()
```



Disscussion and conclusion :

Review of results:
1)As we can see the simplest Linear model cannot even achieve 10% accuracy, while Sequential model is able to achive 12% accuracy. Naive Bayes and SVM models are able to achieve much higher results. This is not a good result, but it is better than random guessing.We can try to improve the accuracy by using more complex models, but it will take a lot of time and resources. We can also try to use transfer learning, but it will also take a lot of time and resources. So, we will use the Sequential model for the final model. We can understand, that complext methods even show effectiveness as this Kazakhs MNIST is, where not over 20 images of each class are available. So, we will use the Sequential model for the final model
Next step :
2) In short, we made a dataset where people drew Kazakh letter with a mouse, we applied different algorithms on the same data to understand which of the algorithms best copes with the classification of letters. The task is difficult since the resolution of the picture is 278*278 and the count of letters are 42. So it is important to find out which algorithms will be used to solve this issue. It might be helpful for people with bad eye vision to understand these letters so in the further usage we now found that sequential model is the best for that work and we can use that for diversity of different projects which may require that ability.

Links of the materials:

1)https://www.youtube.com/w
atch?v=vfyZf2Wj3pU&list=PLA0
M1Bcd0w8ynD1umfubKq1OBY
RXhXkmH

2)https://habr.com/ru/post/50
9472/

3)https://www.javatpoint.com/
machine-learning-naive-bayes-
classifier

4)https://keras.io/guides/seque
ntial_model/