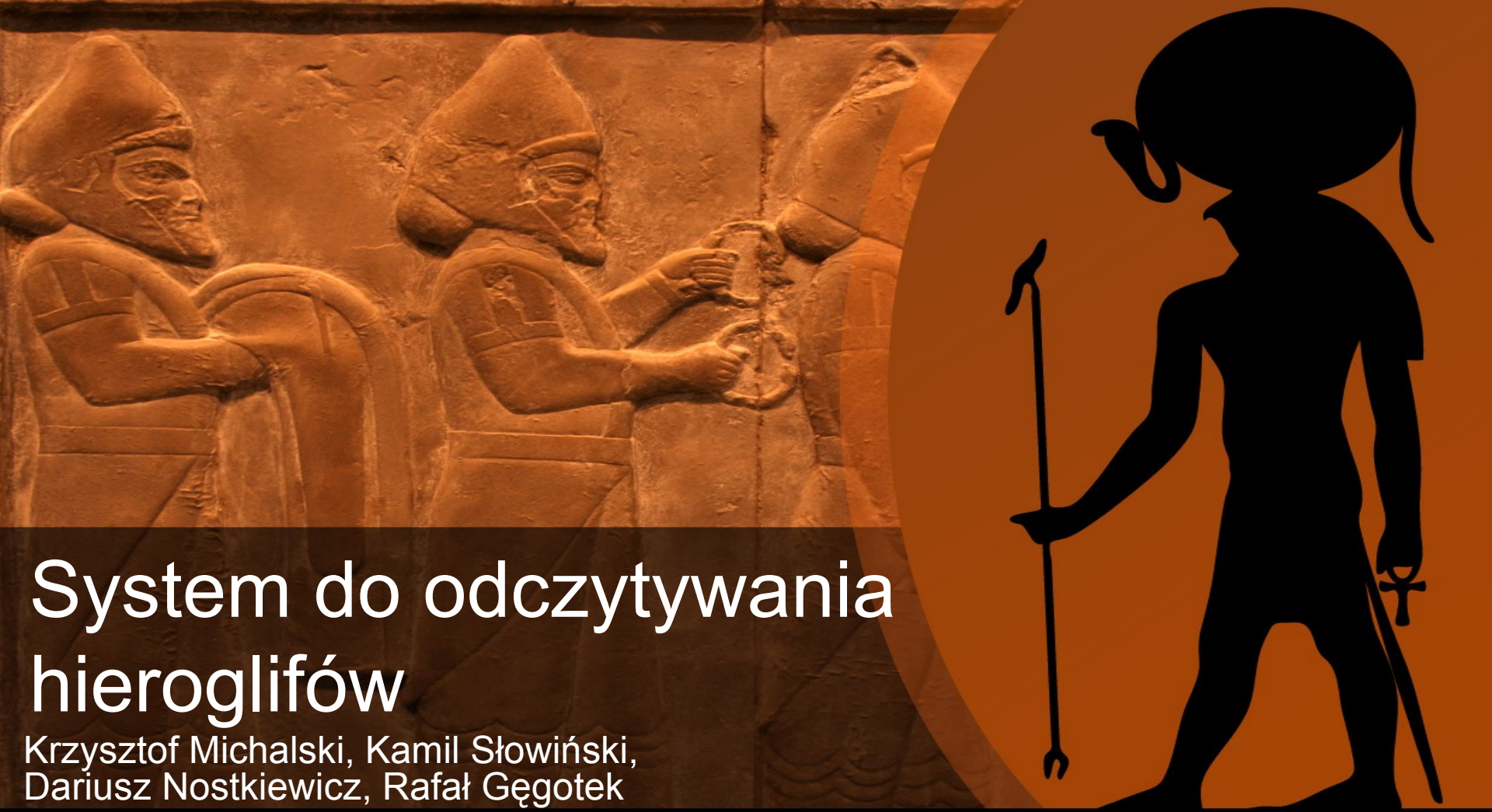
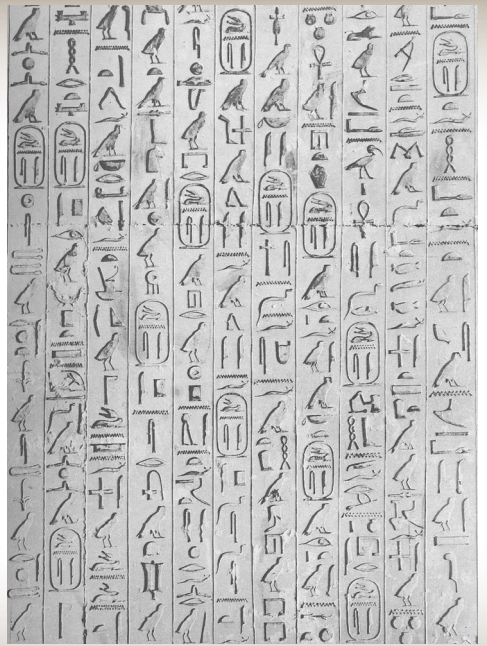


System do odczytywania hieroglifów

Krzysztof Michalski, Kamil Słowiński,
Dariusz Nostkiewicz, Rafał Gęgotek

Cel projektu

Celem projektu było napisanie programu rozpoznającego i tłumaczącego egipskie hieroglify.



Oznaczenia hieroglifów

Do rozpoznawania i przetłumaczenia hieroglifów wykorzystana została lista znaków Gardinera zawierająca około 800 hieroglifów. Każdy hieroglif ma własne oznaczenie składające się z jednej litery i numeru np. B16. Hieroglify podzielone są na kategorie w zależności od tego co przedstawiają. Kategorię w oznaczeniu wskazuje litera alfabetu np. A przedstawiają mężczyznę, B - kobietę, E - ssaki, G - ptaki.

Znaczenie hieroglifów

Każdy znak może oznaczać literę, słowo, lub całe wyrażenie. Często jeden znak ma wiele podobnych znaczeń, jednak dużo hieroglifów do dzisiaj nie zostało przetłumaczonych.

Hieroglyph	Gardiner Unicode	Description	Transliteration	Phonetic	Notes
	A		**Man and his occupations**		
	A1 U+13000	seated man	**I** (masculine) (*p'w*) **male**, **man**, **typical masculine role**, **son**, **courtier** (masculine)		Commonly placed behind a name to indicate masculine sex of named person
	A2 U+13001	man with hand to mouth	**eat** (*wnm*) **drink** (*swr*) **speak**, **think**, **feel**, **tell** (*sgd*) **refrain from speech** (*gr*) **advise/counsel** (*kȝ*) **love** (*mrj*)		Activities involving the mouth, head, or ideas
	A3 U+13002	man sitting on heel	**sit** (*ḥmsi*)		

Biblioteki

- OpenCV - używane do przygotowania obrazu dla sieci neuronowej (konwertowanie obrazu do odpowiedniej przestrzeni barw oraz zmiana rozmiaru obrazu)
- Tensorflow.keras - służący do stworzenia sieci neuronowej rozpoznającej hieroglify
- Numpy - używana do obliczeń numerycznych
- Matplotlib.pyplot - używany do tworzenie wykresów

Dane uczące

Zbiór danych to zbiór stworzony przez Morris Frankena, na podstawie hieroglifów z piramidy Unisa, zawierający w sobie 4210 ręcznie oznaczonych hieroglifów. Zbiór ten posiada 172 klasy.

S29:265, V13:79, G43:197, D21:183, O50:106, X1:232, M23:38, G17:195, Y5:8, N35:448, UNKNOWN:179, D36:59, M1:3, E34:122, M17:364,
D2:24, U33:17, V28:36, N1:18, W25:12, I9:146, N31:17, E23:10, Q3:77, Y2:7, N37:30, H6:7, D58:36, M8:2, G1:35,
D54:12, Q7:3, R8:67, P8:16, I10:41, V25:1, F34:10, G39:21, N16:1, F40:2, W24:39, G14:2, V31:133, N2:1, D35:57,
U1:25, D4:37, G4:8, G25:27, V30:8, Z1:49, F12:1, V22:1, G5:31, F18:6, D46:50, Z7:4, Z11:11, X8:6, F22:1,
M40:3, O28:10, O49:13, O1:20, O4:14, N5:21, D52:5, V24:5, O34:19, Aa26:5, F23:1, U7:4, F13:8, E9:10, M195:3,
D39:2, T22:9, D19:3, N29:18, O31:7, F9:3, M44:7, D56:3, Aa27:3, N41:3, F29:1, S34:10, M3:4, N17:8, D34:1,
T30:3, F21:1, G26:2, M29:3, P1:5, N19:1, D28:17, N18:19, G29:3, P98:5, V6:2, X6:1, F26:4, G36:9, W14:1,
F31:8, N25:2, M18:13, T14:1, U28:3, E1:7, N14:14, R4:3, T21:3, O29:1, U15:13, W18:7, D156:3, M4:1, F35:3,
F30:1, V7:5, V4:13, N24:1, W11:5, G40:8, G50:1, G7:11, T28:2, D1:5, A55:1, Aa15:3, G35:38, G21:2, D60:6,
Q1:17, P13:1, F32:1, W15:1, Y3:3, D10:3, P6:3, M42:5, F16:7, F4:4, D53:1, O11:1, O51:1, G10:1, N26:1,
T20:4, G37:3, N30:12, V16:1, N36:2, M16:2, M26:1, M12:3, D62:1, E17:1, L1:3, W22:2, S24:2, Y1:1, W19:4,
Aa28:1, M20:3, S28:2, U35:1, I5:1, S42:1, M41:3,

Analiza kodu

1.Przetwarzanie obrazu

Funkcja służąca do przekonwertowania obrazu na odcienie szarości i zmiany rozmiaru obrazu:

```
def preprocess_img(img_path, dim=(30, 30)):
    img = cv2.imread(img_path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    img = cv2.resize(img, dim, interpolation=cv2.INTER_LINEAR)
    return img
```

2. Przygotowanie danych uczących

Funkcja odczytująca oznaczenie hieroglifu z nazwy pliku:

```
def get_hiero_class_from_file(file_path):
    dash_index = file_path.index('_')
    return file_path[dash_index+1:].split('.')[0]
```

Wczytanie danych uczących i sprawdzenie liczby obrazów przypadających na jeden hieroglif:

```
files = glob.glob(DATA_PATH + '/*/*', recursive=True

images = []
labels = []
for file in files:
    img = preprocess_img(file)
    label = get_hiero_class_from_file(file)
    images.append(img)
    labels.append(label)
images = np.array(images)
labels = np.array(labels)

all_labels = np.unique(labels)
print(len(all_labels))
print(Counter(labels))
```

2. Przygotowanie danych uczących

Następnym krokiem było powiększenie liczby obrazów uczących, gdyż wiele z klas miało tylko po 1 obrazie w zbiorze.

```
count = Counter(labels)
imgs = []
labs = []
for i, (image, label) in enumerate(zip(images, labels)):
  if count[label] < 40:
    num = count[label]
    while num < 100:
      num+=1
      imgs.append(image)
      labs.append(label)

print(np.shape(images), np.shape(labels))
print(np.shape(imgs), np.shape(labs))
images = np.append(images, imgs)
labels = np.append(labels, labs)
images = images.reshape((98893, 30, 30))
print(np.shape(images), np.shape(labels))
print(Counter(labels))
```

3.Sieć neuronowa

```python
def ATCNet(shape, n_classes):
    # INPUT BLOCK
    input = layers.Input(shape=shape)
    x = layers.Conv2D(64, (3, 3), padding='same', use_bias=False, name='input_block_conv1')(input)
    x = layers.BatchNormalization(name='input_block_conv1_bn')(x)
    x = layers.MaxPooling2D((3, 3), strides=(2,2), padding="same", name = "input_block_conv1_pool")(x)
    x = layers.Activation('relu', name='input_block_conv1_act')(x)
    x = layers.Conv2D(64, (3, 3), padding='same', use_bias=False, name='input_block_conv2')(x)
    x = layers.BatchNormalization(name='input_block_conv2_bn')(x)
    x = layers.MaxPooling2D((3, 3), strides=(2,2), padding="same", name = "input_block_conv2_pool")(x)
    x = layers.Activation('relu', name='input_block_conv2_act')(x)
    # MIDDLE BLOCKS
    # MIDDLE BLOCK 1
    x = layers.SeparableConv2D(128, (3, 3), padding='same', use_bias=False, name='middle_block1_sepconv1')(x)
    x = layers.BatchNormalization(name='middle_block1_sepconv1_bn')(x)
    x = layers.Activation('relu', name='middle_block1_sepconv1_act')(x)
    x = layers.SeparableConv2D(128, (3, 3), padding='same', use_bias=False, name='middle_block1_sepconv2')(x)
    x = layers.BatchNormalization(name='middle_block1_sepconv2_bn')(x)
    x = layers.MaxPooling2D((3, 3), strides=(2,2), padding="same", name='middle_block1_sepconv2_pool')(x)
    x = layers.Activation('relu', name='middle_block1_sepconv2_act')(x)
    # MIDDLE BLOCK 2
    x = layers.SeparableConv2D(128, (3, 3), padding='same', use_bias=False, name='middle_block2_sepconv1')(x)
    x = layers.BatchNormalization(name='middle_block2_sepconv1_bn')(x)
    x = layers.Activation('relu', name='middle_block2_sepconv1_act')(x)
    x = layers.SeparableConv2D(128, (3, 3), padding='same', use_bias=False, name='middle_block2_sepconv2')(x)
    x = layers.BatchNormalization(name='middle_block2_sepconv2_bn')(x)
    x = layers.MaxPooling2D((3, 3), strides=(2,2), padding="same", name='middle_block2_sepconv2_pool')(x)
    x = layers.Activation('relu', name='middle_block2_sepconv2_act')(x)
    # MIDDLE BLOCK 3
    x = layers.SeparableConv2D(256, (3, 3), padding='same', use_bias=False, name='middle_block3_sepconv1')(x)
    x = layers.BatchNormalization(name='middle_block3_sepconv1_bn')(x)
    x = layers.Activation('relu', name='middle_block3_sepconv1_act')(x)
    x = layers.SeparableConv2D(256, (3, 3), padding='same', use_bias=False, name='middle_block3_sepconv2')(x)
    x = layers.BatchNormalization(name='middle_block3_sepconv2_bn')(x)
    x = layers.MaxPooling2D((3, 3), strides=(2,2), padding="same", name='middle_block3_sepconv2_pool')(x)
    x = layers.Activation('relu', name='middle_block3_sepconv2_act')(x)
    # MIDDLE BLOCK 4
    x = layers.SeparableConv2D(256, (3, 3), padding='same', use_bias=False, name='middle_block4_sepconv1')(x)
    x = layers.BatchNormalization(name='middle_block4_sepconv1_bn')(x)
    x = layers.Activation('relu', name='middle_block4_sepconv1_act')(x)
    x = layers.SeparableConv2D(256, (3, 3), padding='same', use_bias=False, name='middle_block4_sepconv2')(x)
    x = layers.BatchNormalization(name='middle_block4_sepconv2_bn')(x)
    x = layers.MaxPooling2D((3, 3), strides=(2,2), padding="same", name='middle_block4_sepconv2_pool')(x)
    x = layers.Activation('relu', name='middle_block4_sepconv2_act')(x)
    # EXIT BLOCK
    # TOP
    x = layers.GlobalAveragePooling2D(name='global_avg_pool')(x)
    x = layers.Dropout(0.15, name="dropout")(x)
    output = layers.Dense(n_classes, activation='softmax', name='predictions', kernel_regularizer=regularizers.l2(0.01))(x)
    #output = Dense(n_classes, activation='softmax', name='predictions')(x)
    model = Model(input, output, name="ATCNet")
    return model
```

3. Sieć neuronowa

```
def own_model(shape, n_classes):
    return Sequential([
        layers.Conv2D(64, (2,2), activation='relu', padding='same', input_shape=shape),
        layers.MaxPool2D(pool_size=2, strides=2),
        layers.Conv2D(32, (2,2), activation='relu', padding='same'),
        layers.MaxPool2D(pool_size=2, strides=2),
        layers.Flatten(),
        layers.Dense(256, activation='relu'),
        layers.Dense(n_classes, activation='softmax')
    ])
```

4.Uczenie sieci neuronowej

Dane zostały podzielone na zbiór treningowy i testowy, następnie posłużyły uczeniu sieci, uzyskując dokładność 99%

```
X = images/255.0
code = {x: i for i,x in enumerate(all_labels)}
y = to_categorical([code[i] for i in labels])
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

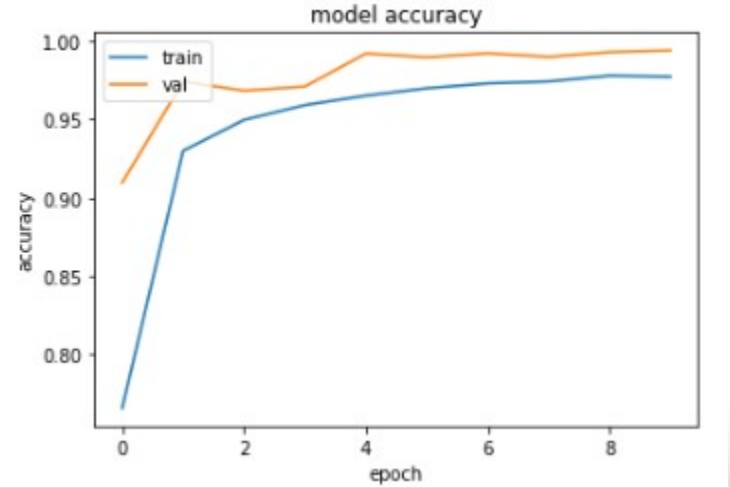
```
model = ATCNet((30, 30, 1), all_labels.shape[0])
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
history = model.fit(X_train, y_train, epochs=10, validation_split=0.1, batch_size=10)
```

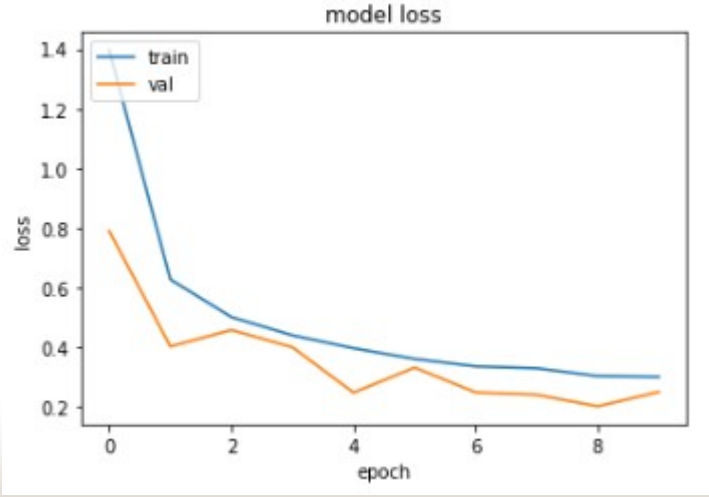
```
model.evaluate(X_test, y_test)
```

```
619/619 [==============================] - 14s 23ms/step - loss: 0.2486 - accuracy: 0.9949
```

4.Uczenie sieci neuronowej

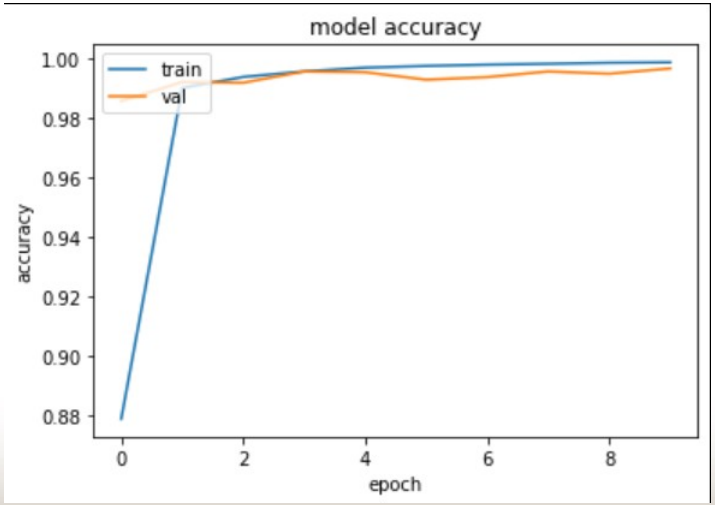
ACTNet

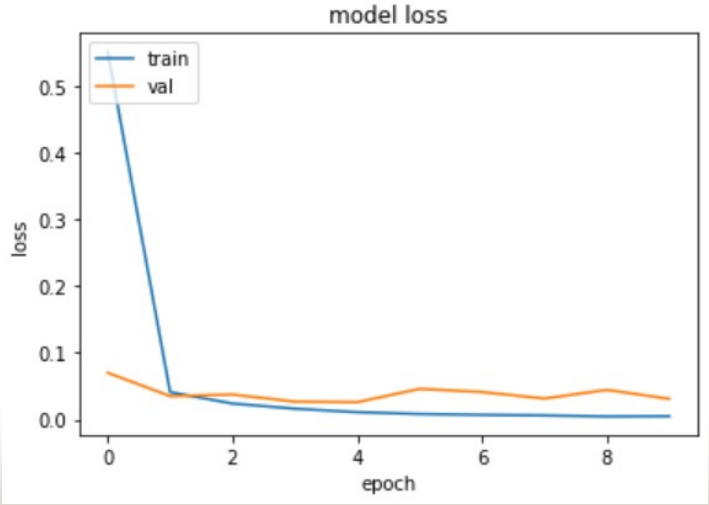




4.Uczenie sieci neuronowej

Własny model sieci



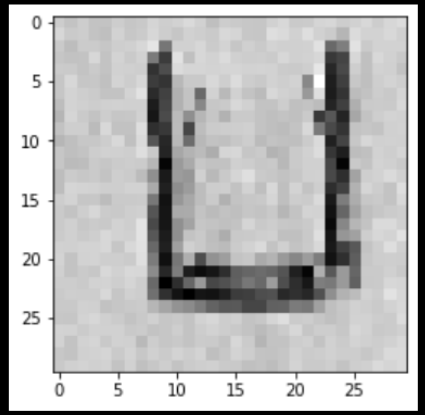


5.Korzystanie z wytrenowanej sieci

Wytrenowana sieć neuronowa pozwala na klasyfikację 172 hieroglifów. Zwraca oznaczenie hieroglifu wg listy znaków Gardinera oraz jego opis i znaczenie po angielsku.

```
model = load_model('model.h5')
img = preprocess_img('xd.png')
plt.imshow(img, cmap='gray')
img = img/255.
img = img.reshape((1, 30, 30, 1))
predictions = model.predict(img)
score = softmax(predictions[0])
idx = np.argpartition(score, -4)[-4:]
idx = idx[np.argsort(np.array(-score)[idx])]
print(
    "This image most likely belongs to {} with a {:.2f} percent confidence."
    .format(all_labels[idx[0]], 100 * score[idx[0]])
)
print_translation(translation.loc()[idx[0]])
```

```
1/1 [==============================] - 0s 270ms/step
This image most likely belongs to D28 with a 1.50 percent confidence.
Symbol ␣ Code D28U+13093
two arms upraised
translation:
closethe Ka (life spirit) (k$^{i}$) phonetic: k$^{i}$(bil.)
note:
```



Podsumowanie

Udało się stworzyć system rozpoznający i tłumaczący pojedyncze hieroglify - cel projektu został osiągnięty.