Neurónové siete – projekt EEG dataset

(riešenie problému klasifikácie/predikcie signálov pomocou hlbokých neurónových sietí)

Riešiteľ: František Kurimský 1Im
Ústav informatiky PF UPJŠ
Rok: 2022/2023 zimný semester

Formulácia problému:

Úlohou projektu je pripraviť modely na klasifikáciu cifier. Jeden z modelov pracuje na EEG dát z datasetu MindBigData, The Visual "MNIST" of Brain Digits (2021). Druhý z modelov pracuje na obrázkoch z toho istého datasetu. Klasifikácia cifier z datasetu obrázkov je dlhšie známy problém, ktorý pracuje s veľmi dobrou presnosťou. Avšak klasifikácia cifier nad EEG dátami je ťažší problém. Po skúmaní podobných prác sa nepodarilo nájsť model, ktorý pracuje s dostatočnou presnosťou. Súčasťou projektu je overenie získaných informácií.

Príprava dát:

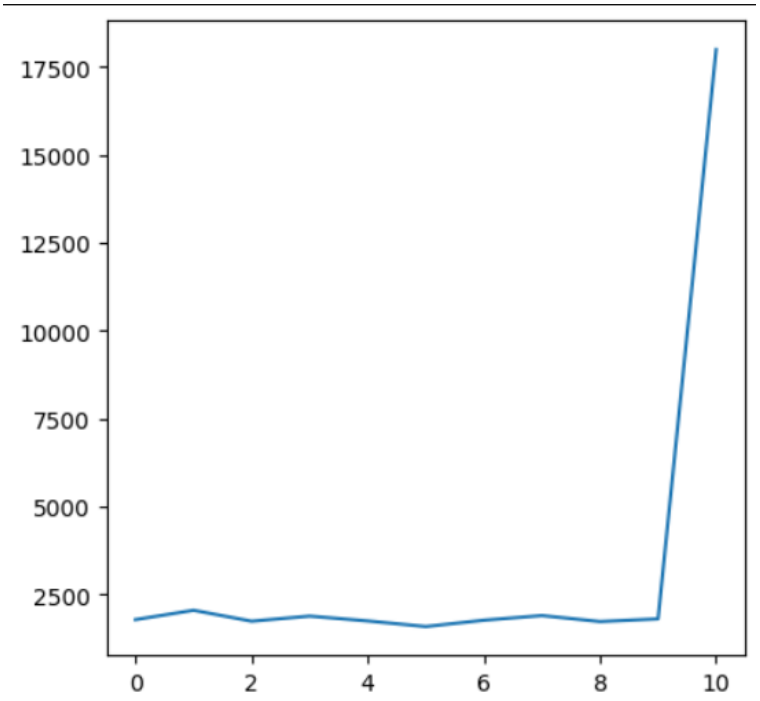
V tomto projekte sa pracuje s datasetom MindBigData, The Visual "MNIST" of Brain Digits (2021) dostupnom na odkaze:
https://vivancos.net/opendb/MindBigDataVisualMnist2021-Muse2v0.17.zip .

Dataset obsahuje csv súbor, v ktorom je 30000 riadkov. Každý riadok sa skladá z týchto hodnôt:

- Dataset : ohodnotenie riadku na TRAIN a TEST, string hodnota
- Origin: integer hodnota ukazujúca na pôvodnú lokalitu v datasete Yann LeCun MNIST
- Digit_event: integer hodnota číslice od 0-9, pre iadok kde nenastáva udalosť číslice je hodnota -1
- Original_png: 784 integer hodnôt reprezentujúce pôvodný obrázok o rozmere 28 na 28 pixelov
- Timestamp: unixový timestamp hovoriaci o počiatku aktuálnej udalosti
- EEGdata: obsahuje 4 * 512 floating hodnôt, kde:
    - 512 prvých hodnôt reprezentujú dáta pre TP9 senzor vo frekvencii 256 hz
    - 512 druhých hodnôt reprezentujú dáta pre AF7 senzor vo frekvencii 256 hz
    - 512 tretích hodnôt reprezentujú dáta pre AF8 senzor vo frekvencii 256 hz
    - 512 štvrtých hodnôt reprezentujú dáta pre TP10 senzor vo frekvencii 256 hz

V tomto projekte s ďalšími hodnotami v riadkoch nepracujeme.

Pri spracovaní dát som prepísal hodnoty -1 v stĺpci Digit_event na 10. Počty udalostí s jednotlivými ciframi su ukázané v nasledujúcom obrázku.



Obrázok 1 Distribúcia udalostí

Predspracovanie dát:

Predspracovanie dát pre model klasifikácie z obrázkov bol nasledujúci.

Pripravil som si dataframe obsahujúci dáta obrázkov. Pre predstavu, ako vyzerá obrázok som si vytvoril podmnožinu obrázkov, kde sa nenachádza žiadny obrázok, ktorý pozostáva len z núl.

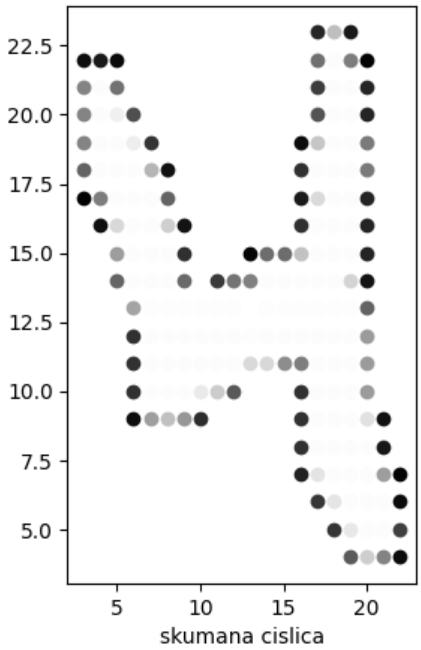
Následne som si vybral jeden riadok, aby som simuloval správanie generátora. Hodnoty tohto riadku som reshapol na tvar 28 krát 28 pixelov. Pomocou matplotlib knižnice som si danú číslicu zobrazil.

Kód pre spracovanie dát obrázkov je nasledujúci.

```
imgs = df.iloc[:,3:784+3]
cc = imgs.loc[imgs[imgs > 0].any(axis=1)]
cc1 = cc.iloc[20,:]
cc2 = np.array(cc1)
cc2=np.reshape(cc2,(28,28), 'A')
print(np.amax(cc2))

plt.figure(figsize = (3, 5))
for i in range(28):
    for j in range(28):
        if cc2[i,j]>0:
            # print(str(cc2[i,j]/255))
            plt.plot(j,27-i, "o", color=str(cc2[i,j]/255), markersize=6)
plt.xlabel('skumana cislica')
plt.show()
```

Skúmaná číslica vyzerá takto:



Obrázok 2 Skúmaná číslica

Spracovanie dát pre model pracujúci nad množinou EEG dát je nasledujúci.

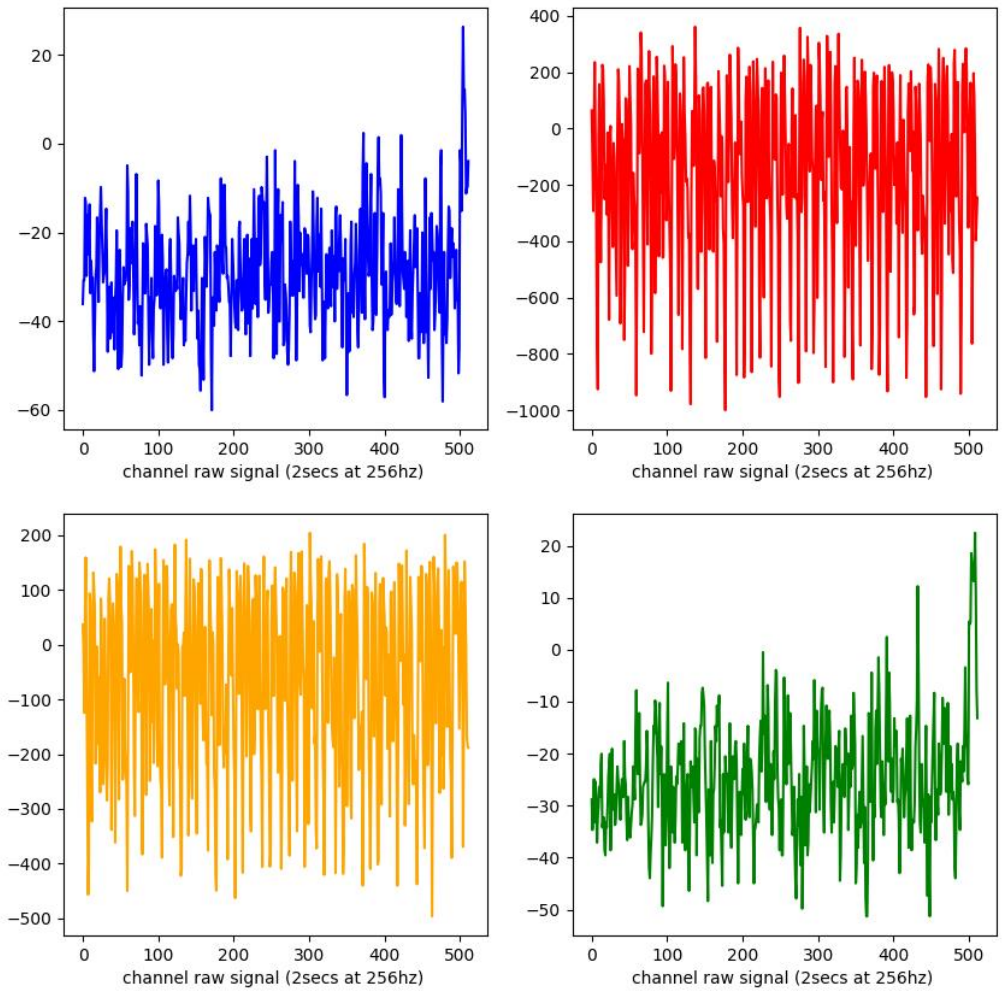
Najprv som si pripravil dataframe, ktorý obsahoval len EEG dáta z pôvodného datasetu. Následne som si z nového datafremu vybral jeden riadok, aby som simuloval, ako bude prebiehať príprava dát v generátore pre model neurónovej siete. Daný riadok som rozdelil na 4 časti, z ktorej každá predstavuje hodnoty z jedného EEG senzora. Rozdelenie bolo popísané v časti prípravy dát. Tieto hodnoty som si pomocou matplotlib knižnice zobrazil na časovej osi. Kód tejto časti je nasledujúci.

```
posun = 4 + 784
finish = 4 * 512
eeg_data = df.iloc[:, posun:posun+finish]
aa = eeg_data.iloc[20, :]
bb1=aa[0:512]
bb1 = bb1.reset_index(drop=True)
bb2=aa[512:512*2]
bb2 = bb2.reset_index(drop=True)
bb3=aa[512*2:512*3]
bb3 = bb3.reset_index(drop=True)
bb4=aa[512*3:512*4]
bb4 = bb4.reset_index(drop=True)

plt.figure(figsize =(10,10))
plt.subplot(2,2,1)
plt.xlabel('channel raw signal (2secs at 256hz) ')
plt.plot(bb1, color='blue')
```

```
plt.subplot(2,2,2)
plt.xlabel('channel raw signal (2secs at 256hz) ')
plt.plot(bb2, color='red')
plt.subplot(2,2,3)
plt.xlabel('channel raw signal (2secs at 256hz) ')
plt.plot(bb3, color='orange')
plt.subplot(2,2,4)
plt.xlabel('channel raw signal (2secs at 256hz) ')
plt.plot(bb4, color='green')
plt.savefig('assets/eeg_signals_4_channels_row_20.png')
```

Výsledný obrázok tejto časti je nasledujúci.



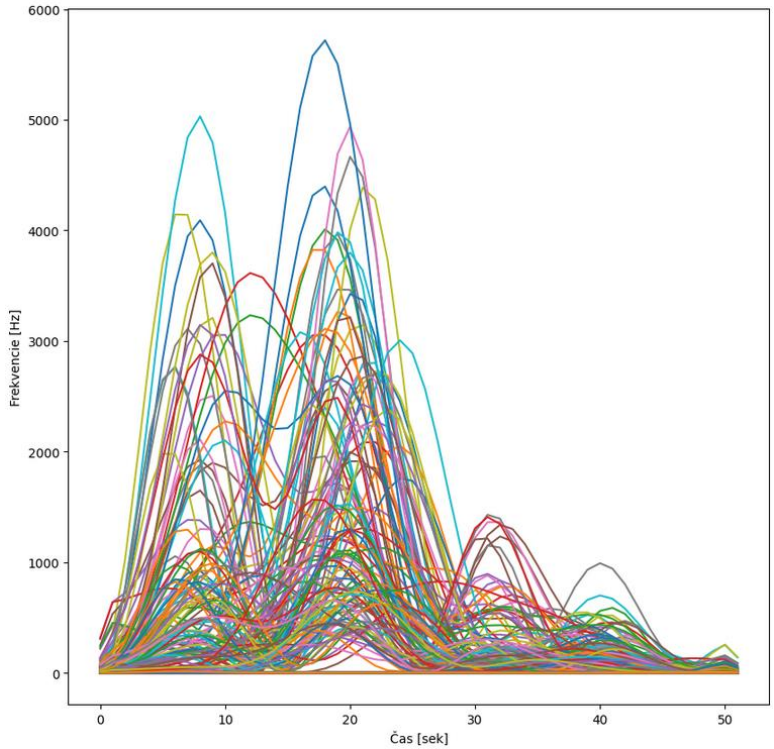
Obrázok 3 EEG dáta jedného riadku zo 4 senzorov

Ďalším krokom spracovania týchto dát bolo príprava spectogramu. Pre získanie spectogramov som použil knižnicu „scipy.signal.spectrogram“. Pre každú zo štyroch častí som pripravil spectogram, a nakoniec som tieto spectogramy spojil do jedného. Kód pre túto časť je nasledujúci.

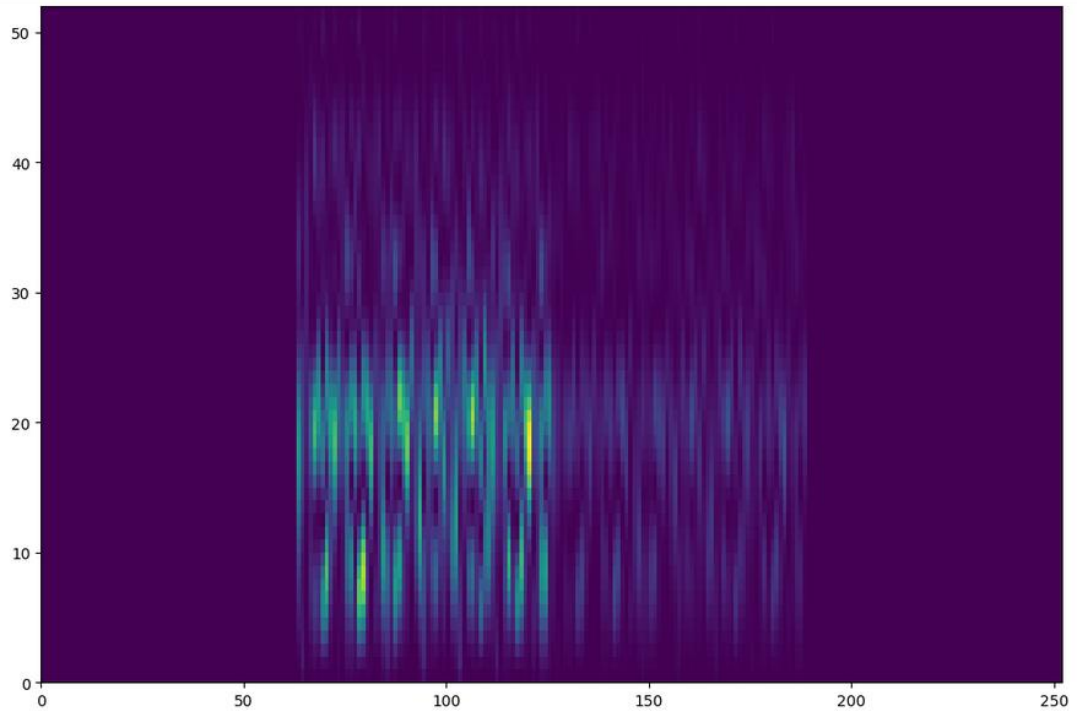
```
f,t,Sxx = scipy.signal.spectrogram(bb1, 256,nperseg=13,noverlap=5,nfft=102)
f2,t2,Sxx2 = scipy.signal.spectrogram(bb2, 256,nperseg=13,noverlap=5,nfft=102)
f3,t3,Sxx3 = scipy.signal.spectrogram(bb3, 256,nperseg=13,noverlap=5,nfft=102)
f4,t4,Sxx4 = scipy.signal.spectrogram(bb4, 256,nperseg=13,noverlap=5,nfft=102)

y_shape = 63
final=np.zeros((52,y_shape*4))
final[0:52,0:y_shape]=Sxx
final[0:52,y_shape:y_shape*2]=Sxx2
final[0:52,y_shape*2:y_shape*3]=Sxx3
final[0:52,y_shape*3:y_shape*4]=Sxx4
print('vystupyShape=',final)
fig2=plt.figure(figsize=(12,8))
plt.pcolormesh(final)
plt.show()
plt.plot(final)
plt.ylabel('Frekvencie [Hz]')
plt.xlabel('Čas [sek]')
plt.show()
```

Výstupom tejto časti boli nasledujúce obrázky. *Obrázok 4.* ukazuje získané dáta v závislosti času od frekvencie. *Obrázok 5.* ukazuje výsledný spectrogram.



Obrázok 4 Spectogram jedného riadku plain



Obrázok 5 Spectogram jedného riadku

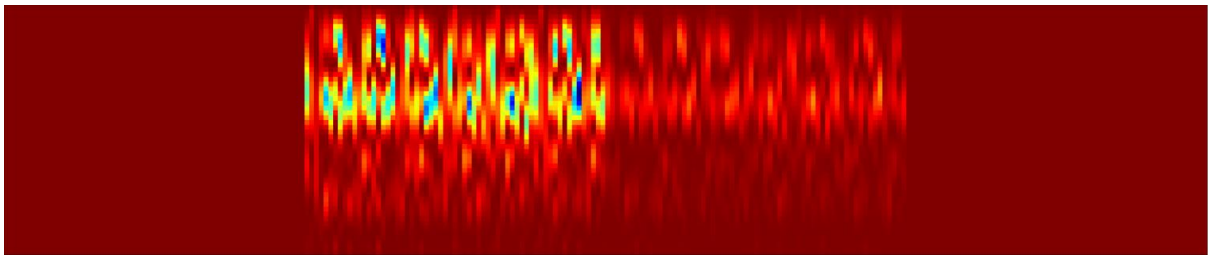
Pri trénovaní modelu som priamo s dátami získanými z knižnice scipy, ale takisto som pomocou knižnice cv2 pripravil reprezentáciu spectogramu do rgb podoby, respektíve do colormap podoby, aby som tieto dáta vedel použiť pri trénovaní pripravených modelov. Pre túto reprezentáciu som použil COLORMAP_JET z knižnice cv2. Nasledujúci kód ukazuje prípravu tejto reprezentácie.

```
# Normalize data
data = (final - np.min(final)) / (np.max(final) - np.min(final))

# Create a grayscale image
gray_image = np.uint8(data * 255)

# Apply a colormap to the grayscale image
colormap = cv2.COLORMAP_JET
color_image = cv2.applyColorMap(gray_image, colormap)
plt.figure(figsize =(50,30))
plt.imshow(color_image)
```

Výsledkom tohto kódu je nasledujúci obrázok.



Obrázok 6 Spectogram pomocou cv2 knižnice v COLORMAP_JET

Príprava modelov neurónovej siete:

Pripravil som model na klasifikáciu cifier nad dátami obsahujúcimi obrázky cifier. Pred modelom som upravil dáta a to rozdelením na tréningovú, testovaciu a validačnú sadu. Tréningová sada obsahuje 80 percent pôvodného datasetu, validačná 20 percent. Pre Testovaciu sadu som sa rozhodol vybrať náhodne 50 percent pôvodného datasetu. Kód pre rozdelenie dát je nasledujúci.

```
import random
labels = df.iloc[:, 2]
length = len(imgs)
indexes = np.arange(0, len(imgs), 1)
indexes
train_length = int(length * 0.8)
test_length = int(length * 0.5)
train_indexes = np.random.choice(train_length, train_length, replace=False)
valid_indexes = [x for x in range(len(imgs)) if x not in train_indexes]
print('train: ' + str(len(train_indexes)) + ' test: ' + str(len(valid_indexes)))
test_indexes = np.random.choice(test_length, test_length, replace=False)
imgs = np.array(imgs)
eeg_data = np.array(eeg_data)
labels = np.array(labels)
print(len(eeg_data), len(imgs))
train_data_img = imgs[train_indexes]
train_labels_img = labels[train_indexes]
valid_data_img = imgs[test_indexes]
valid_labels_img = labels[test_indexes]
test_data_img = imgs[valid_indexes]
test_labels_img = labels[valid_indexes]
```

Následne som si pomocou knižnice „collections“ zitil váhy pre dané triedy z trénovacej sady. Táto časť je potrebná pre neurónovú sieť, aby lepšie dokázala optimalizovať svoje váhy.

```
from collections import Counter
itemCt = Counter(train_labels_img)
maxCt = float(max(itemCt.values()))
cw = {clsID : maxCt/numImg for clsID, numImg in itemCt.items()}
print(cw)
```

Výstupom tejto častí je množina tried s váhami.

```
{8: 10.50328227571116, 1: 8.894379246448425, 4: 10.23454157782516, 10: 1.0,
7: 9.424083769633508, 9: 10.020876826722338, 3: 9.632107023411372, 2:
10.22001419446416, 6: 10.098176718092567, 0: 9.965397923875432, 5:
11.401425178147269}
```

Pri trénovaní modelu je potrebné si pripraviť generátor, ktorý pred spracováva hodnoty tak, aby boli prispôsobené pre vstupnú vrstvu neurónovej siete. Generátor pracuje podobne, ako sme ukazovali v časti predspracovania dát. Je nasledujúci.

```
import math
class GeneratorImages(tf.keras.utils.Sequence ):

    def __init__(self, paths, labels, batch_size):
        self.paths, self.labels, self.batch_size = paths, labels, batch_size
    def __len__(self):
        return math.ceil(len(self.paths) / self.batch_size)

    def __getitem__(self, idx):
        batch_x = self.paths[idx * self.batch_size : (idx + 1) * self.batch_size]
        batch_y = self.labels[idx * self.batch_size : (idx + 1) * self.batch_size]

        batch_x = [np.reshape(x,(28,28), 'A') for x in batch_x]
        batch_y = tf.keras.utils.to_categorical(batch_y, num_classes=11)

        return np.array(batch_x), np.array(batch_y)



train_gen_img = GeneratorImages(train_data_img, train_labels_img, 128)
test_gen_img = GeneratorImages(test_data_img, test_labels_img, 128)
valid_gen_img = GeneratorImages(valid_data_img, valid_labels_img, 128)
print('done')
```

Generátor nám takto pripraví tri generácie pre trénovanie, validáciu, a testovanie modelu neurónovej siete.

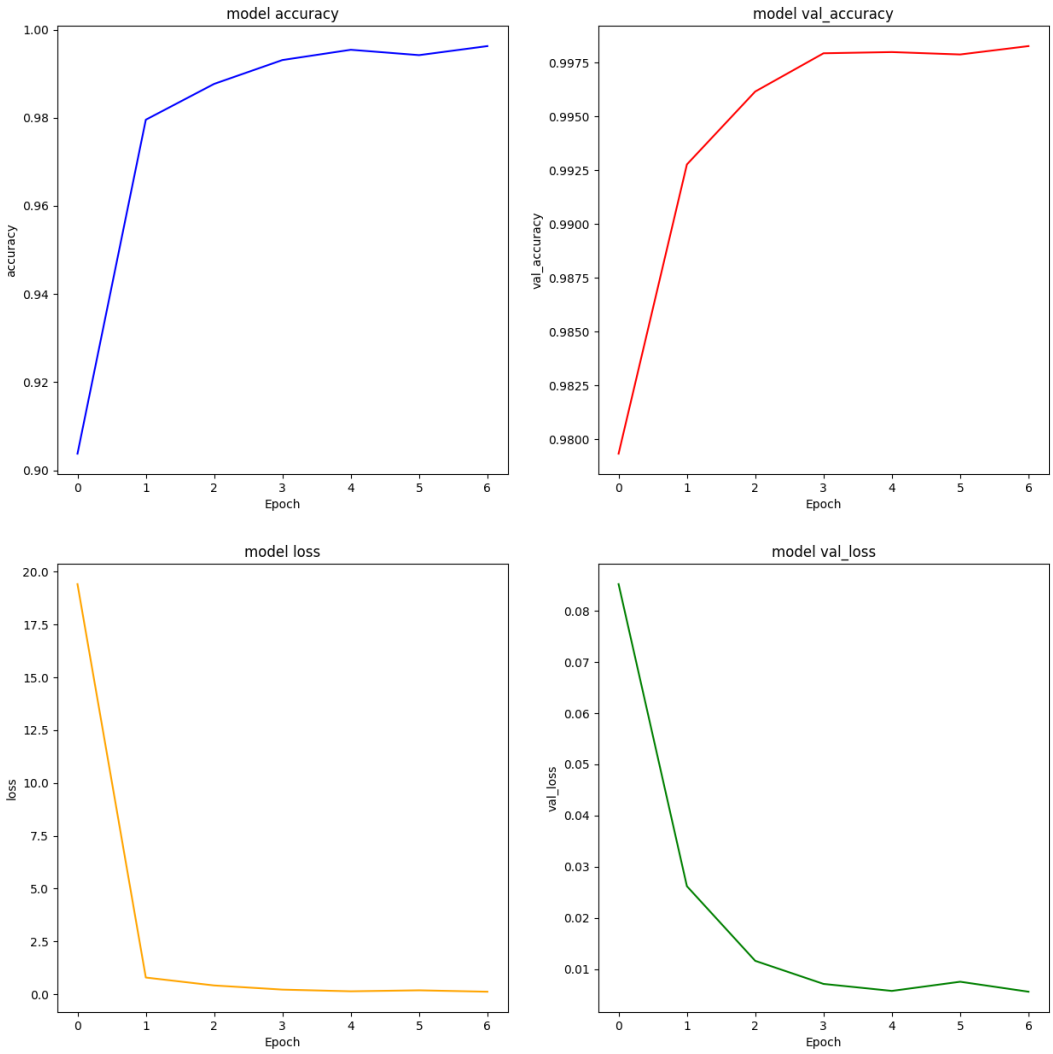
V tomto kroku máme pripravené veci pre trénovanie.

Navrhol som model konvolučnej neurónovej siete pre klasifikáciu nad obrázkami.

```
from tensorflow.keras import layers, models
numberModel = models.Sequential()

numberModel.add(layers.Conv2D(64, (3, 3), input_shape=(28, 28, 1),
activation='relu'))
numberModel.add(layers.MaxPooling2D())
numberModel.add(layers.Dropout(0.2))
numberModel.add(layers.Flatten())
numberModel.add(layers.Dense(128, activation='relu'))
numberModel.add(layers.Dense(11, activation='softmax'))
numberModel.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
```

Model používa 64 filtrov v konvolučnej vrstve. Loss funkciu používame categorical_crossentropy a optimizer adam. Počet epoch trénovania je 7. História trénovania tohto modelu je nasledujúca.



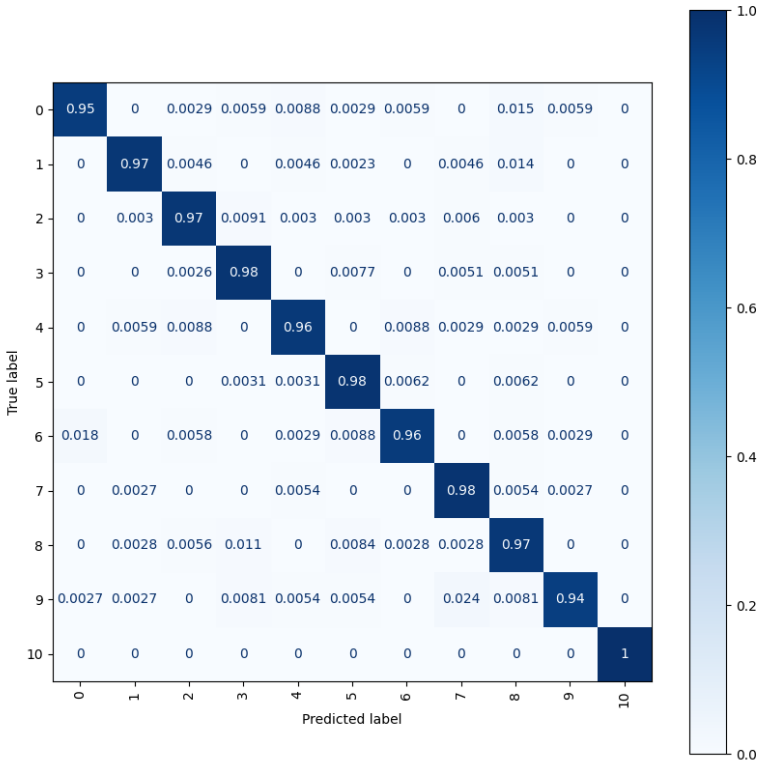
Obrázok 7 História trénovania CNN nad obrázkami

Hodnoty precision, recall, f1 a accuracz boli nasledujúce.

	precision	recall	f1-score	support
0	0.98	0.95	0.97	340
1	0.99	0.97	0.98	432
2	0.97	0.97	0.97	331
3	0.97	0.98	0.97	390
4	0.96	0.96	0.96	340
5	0.96	0.98	0.97	325
6	0.97	0.96	0.96	342
7	0.96	0.98	0.97	372
8	0.94	0.97	0.95	359
9	0.98	0.94	0.96	369
10	1.00	1.00	1.00	3600
accuracy			0.98	7200
macro avg	0.97	0.97	0.97	7200
weighted avg	0.98	0.98	0.98	7200

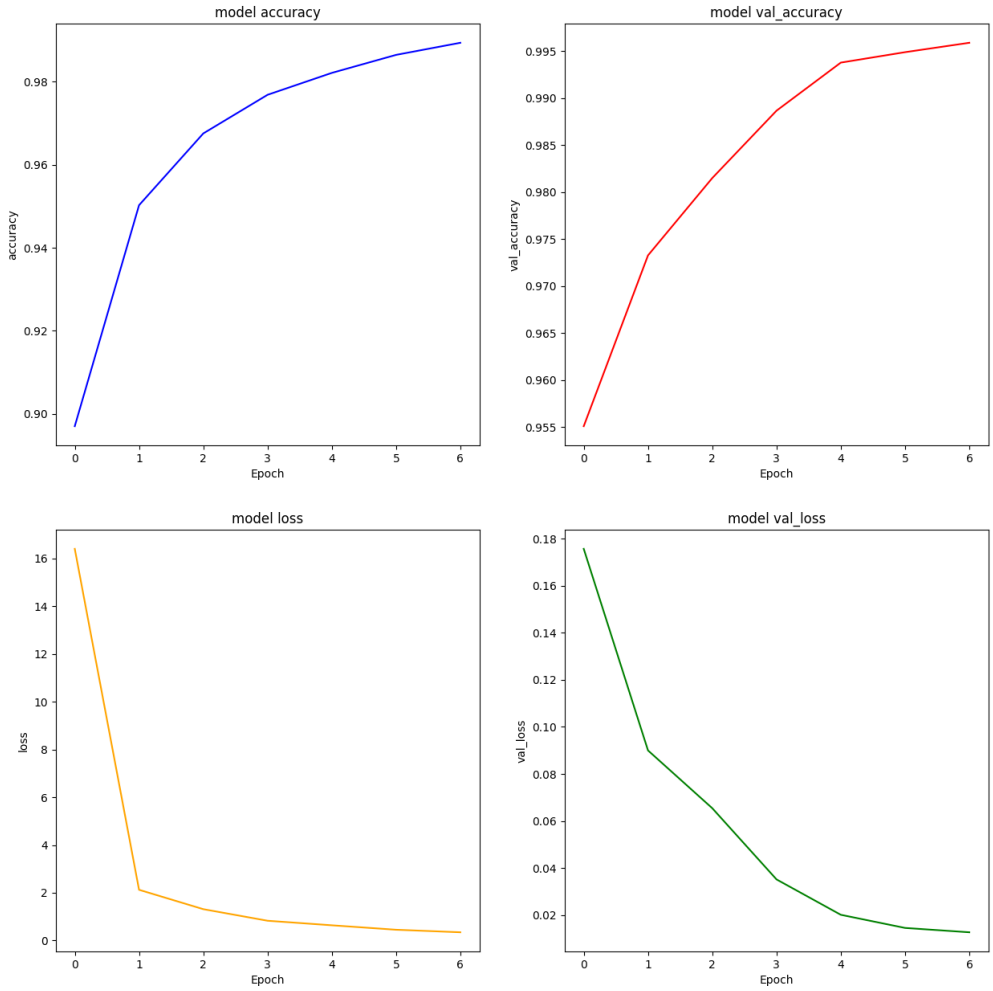
Obrázok 8 Výsledky testovania

Confusion matica je nasledujúca.



Obrázok 9 Confusion matica prvého modelu

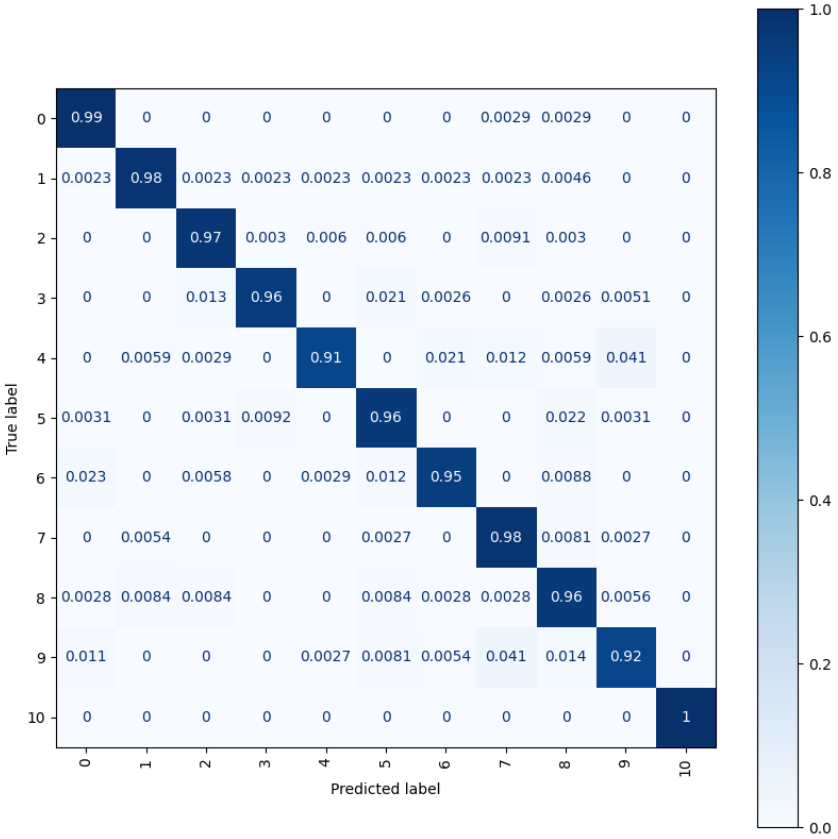
Po takýchto výsledkoch som skúsil znížiť počet filtrov pre konvolučnú vrstvu na 16. Aj pri takto zníženom počte boli výsledky veľmi dobré.



Obrázok 10 História trénovania menší počet filtrov

	precision	recall	f1-score	support
0	0.96	0.99	0.98	340
1	0.98	0.98	0.98	432
2	0.96	0.97	0.97	331
3	0.99	0.96	0.97	390
4	0.98	0.91	0.95	340
5	0.93	0.96	0.95	325
6	0.96	0.95	0.96	342
7	0.94	0.98	0.96	372
8	0.93	0.96	0.95	359
9	0.94	0.92	0.93	369
10	1.00	1.00	1.00	3600
accuracy			0.98	7200
macro avg	0.96	0.96	0.96	7200
weighted avg	0.98	0.98	0.98	7200

Obrázok 11 Precision, recall f1 a accuracy pre znížený počet filtrov



Obrázok 12 Confusion matica pre znížený počet filtrov

Príprava druhého modelu prestavovala rozdelenie dát na trénovaciu, testvaciu a validačnú sadu.

```
import random
labels = df.iloc[:, 2]
length = len(eeg_data)
indexes = np.arange(0, len(eeg_data), 1)
indexes
train_length = int(length * 0.8)
test_length = int(length * 0.5)

train_indexes = np.random.choice(train_length, train_length, replace=False)
valid_indexes = [x for x in range(len(eeg_data)) if x not in train_indexes]
print('train: ' + str(len(train_indexes)) + ' test: ' + str(len(valid_indexes)))

test_indexes = np.random.choice(test_length, test_length, replace=False)
eeg_data = np.array(eeg_data)

train_data_eeg = eeg_data[train_indexes]
train_labels_eeg = labels[train_indexes]
valid_data_eeg = eeg_data[test_indexes]
valid_labels_eeg = labels[test_indexes]
test_data_eeg = eeg_data[valid_indexes]
test_labels_eeg = labels[valid_indexes]
```

Podobne sme si pripravovali dáta, kde sme nemali riadky nepredstavujúce cifry.

```
df_just_numbers = df_just_numbers[df_just_numbers[2] != 10]
eeg_data_just_numbers = df_just_numbers.iloc[:, posun:posun+finish]
```

Pre model sme si pripravili dva typy generátorov, kde jeden dával na výstupe generáciu v colormap podobe, a druhý spectrogram v “čistej” podobe.

Prvý generátor:

```
import math
class GeneratorEeg(tf.keras.utils.Sequence ):

    def __init__(self, paths, labels, batch_size):
        self.paths, self.labels, self.batch_size = paths, labels, batch_size
    def __len__(self):
        return math.ceil(len(self.paths) / self.batch_size)

    def __getitem__(self, idx):
        batch_x = self.paths[idx * self.batch_size : (idx + 1) * self.batch_size]
        batch_y = self.labels[idx * self.batch_size : (idx + 1) * self.batch_size]
```

```
        batch_x = [self.getDeepSpectro(x).reshape(52,63*4,3) for x in batch_x]
        batch_y = tf.keras.utils.to_categorical(batch_y, num_classes=11)

        return np.array(batch_x), np.array(batch_y)

    def getDeepSpectro(self, x):
        bb1=x[0:512]
        bb2=x[512:512*2]
        bb3=x[512*2:512*3]
        bb4=x[512*3:512*4]

        f,t,Sxx = scipy.signal.spectrogram(bb1, 256,nperseg=13,noverlap=5,nfft=102)
        f2,t2,Sxx2 = scipy.signal.spectrogram(bb2, 256,nperseg=13,noverlap=5,nfft=102)
        f3,t3,Sxx3 = scipy.signal.spectrogram(bb3, 256,nperseg=13,noverlap=5,nfft=102)
        f4,t4,Sxx4 = scipy.signal.spectrogram(bb4, 256,nperseg=13,noverlap=5,nfft=102)



        SxxFinal=np.zeros((52,63*4))
        SxxFinal[0:52,0:63]=Sxx
        SxxFinal[0:52,63:63*2]=Sxx2
        SxxFinal[0:52,63*2:63*3]=Sxx3
        SxxFinal[0:52,63*3:63*4]=Sxx4
        SxxFinal = (SxxFinal - np.min(SxxFinal)) / (np.max(SxxFinal) - np.min(SxxFinal))
        gray_image = np.uint8(SxxFinal * 255)

        # Apply a colormap to the grayscale image
        colormap = cv2.COLORMAP_JET
        color_image = cv2.applyColorMap(gray_image, colormap)

        return(color_image)

train_gen_eeg = GeneratorEeg(train_data_eeg, train_labels_eeg, 128)
test_gen_eeg = GeneratorEeg(test_data_eeg, test_labels_eeg, 128)
valid_gen_eeg = GeneratorEeg(valid_data_eeg, valid_labels_eeg, 128)
print(train_labels_eeg)
```

Druhý generátor:

```
import math
class GeneratorEeg(tf.keras.utils.Sequence ):

    def __init__(self, paths, labels, batch_size):
        self.paths, self.labels, self.batch_size = paths, labels, batch_size
    def __len__(self):
        return math.ceil(len(self.paths) / self.batch_size)

    def __getitem__(self, idx):
        batch_x = self.paths[idx * self.batch_size : (idx + 1) * self.batch_size]
```

```
        batch_y = self.labels[idx * self.batch_size : (idx + 1) * self.batch_size]


        batch_x = [self.getDeepSpectro(x).reshape(52,63*4,3) for x in batch_x]
        batch_y = tf.keras.utils.to_categorical(batch_y, num_classes=11)

        return np.array(batch_x), np.array(batch_y)

    def getDeepSpectro(self, x):
        bb1=x[0:512]
        bb2=x[512:512*2]
        bb3=x[512*2:512*3]
        bb4=x[512*3:512*4]

        f,t,Sxx = scipy.signal.spectrogram(bb1, 256,nperseg=13,noverlap=5,nfft=102)
        f2,t2,Sxx2 = scipy.signal.spectrogram(bb2, 256,nperseg=13,noverlap=5,nfft=102)
        f3,t3,Sxx3 = scipy.signal.spectrogram(bb3, 256,nperseg=13,noverlap=5,nfft=102)
        f4,t4,Sxx4 = scipy.signal.spectrogram(bb4, 256,nperseg=13,noverlap=5,nfft=102)


        SxxFinal=np.zeros((52,63*4))
        SxxFinal[0:52,0:63]=Sxx
        SxxFinal[0:52,63:63*2]=Sxx2
        SxxFinal[0:52,63*2:63*3]=Sxx3
        SxxFinal[0:52,63*3:63*4]=Sxx4

        return(SxxFinal)

train_gen_eeg = GeneratorEeg(train_data_eeg, train_labels_eeg, 128)
test_gen_eeg = GeneratorEeg(test_data_eeg, test_labels_eeg, 128)
valid_gen_eeg = GeneratorEeg(valid_data_eeg, valid_labels_eeg, 128)
print(train_labels_eeg)
```

Tieto generátory predstavujú časť spracovania dát z časti vyššie.

Modely, and ktorými sme trénovali sú nasledujúce.

Prvý model:

```
from tensorflow.keras import layers, models
eegModel = models.Sequential()

eegModel.add(layers.Conv2D(32, (3, 3), input_shape=(52, 63*4, 3), activation='relu'))
eegModel.add(layers.MaxPooling2D())
eegModel.add(layers.Conv2D(32, (3, 3), activation='relu'))
eegModel.add(layers.MaxPooling2D())
eegModel.add(layers.Conv2D(32, (3, 3), activation='relu'))
eegModel.add(layers.MaxPooling2D())
eegModel.add(layers.Dropout(0.2))
```

```
eegModel.add(layers.Flatten())
eegModel.add(layers.Dense(128, activation='relu'))
eegModel.add(layers.Dense(11, activation='softmax'))
eegModel.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

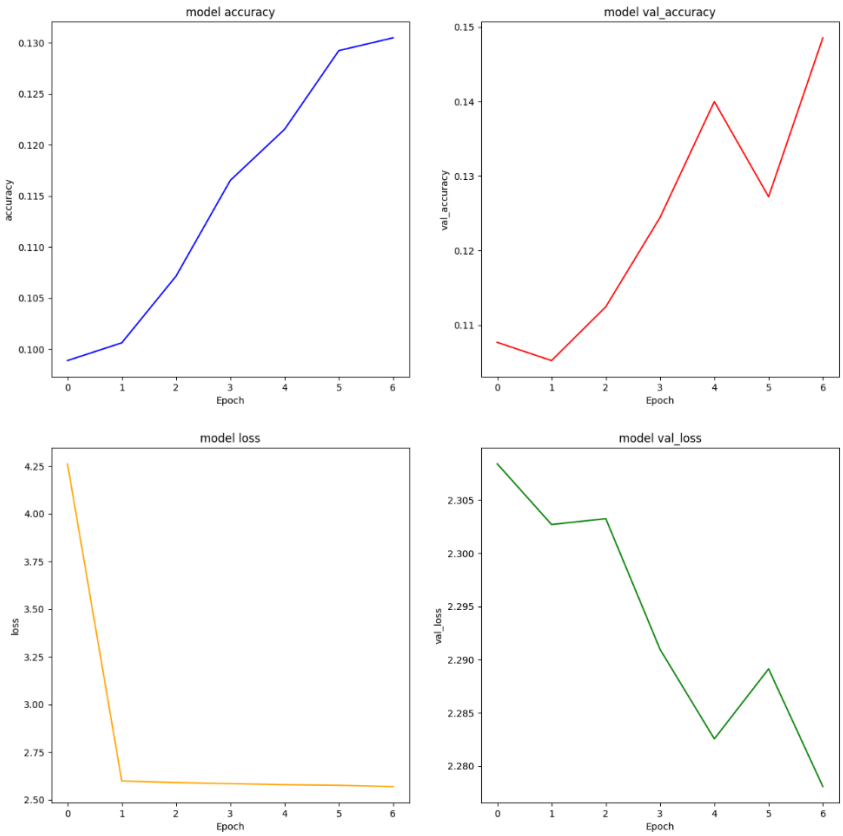
Druhý model:

```
from tensorflow.keras import layers, models
eegModel = models.Sequential()

pretrained_model= tf.keras.applications.ResNet50(include_top=False,
                   input_shape=(52, 44, 3),
                   pooling='avg',classes=11,
                   weights='imagenet')
# for layer in pretrained_model.layers:
#         layer.trainable=False

eegModel.add(pretrained_model)
eegModel.add(layers.Flatten())
eegModel.add(layers.Dense(128, activation='relu'))
eegModel.add(layers.Dense(11, activation='softmax'))
```

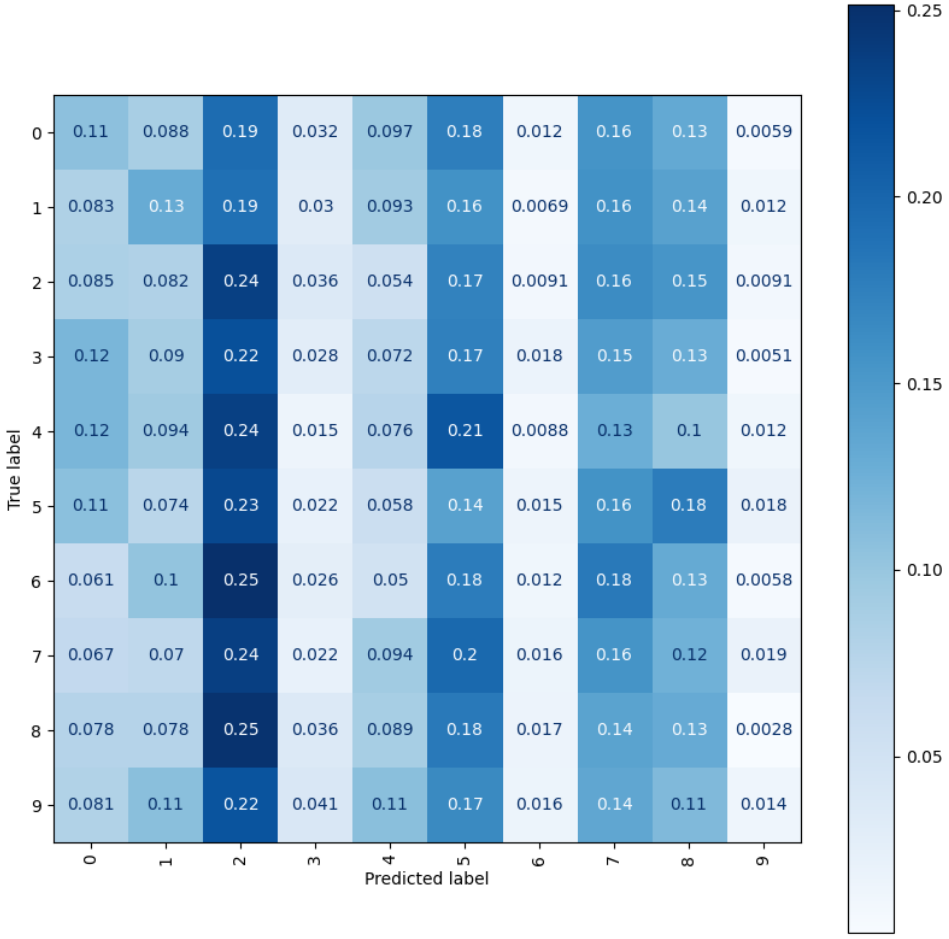
Nasledujúca časť obsahuje výsledky prvej neurónovej siete na datasete obsahujúcom len udalosti s cifrou s prvým generátorom.



Obrázok 13 História trénovania EEG 1. model 1. generátor

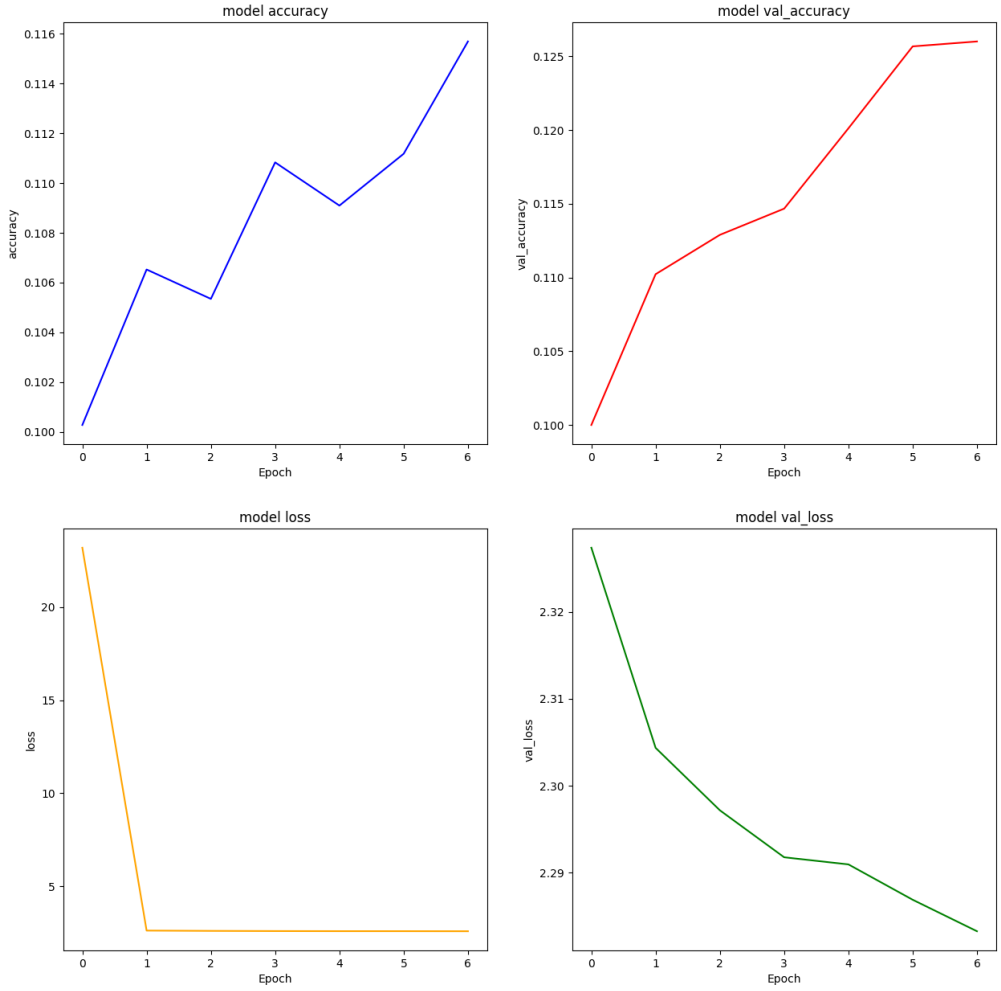
	precision	recall	f1-score	support
0	0.11	0.11	0.11	340
1	0.17	0.13	0.15	432
2	0.10	0.24	0.14	331
3	0.11	0.03	0.04	390
4	0.09	0.08	0.08	340
5	0.07	0.14	0.10	325
6	0.09	0.01	0.02	342
7	0.11	0.16	0.13	372
8	0.10	0.13	0.11	359
9	0.14	0.01	0.02	369
accuracy			0.10	3600
macro avg	0.11	0.10	0.09	3600
weighted avg	0.11	0.10	0.09	3600

Obrázok 14 Hodnoty precision, recall f1 a accuracz nad EEG pre 1. model 1. generátor



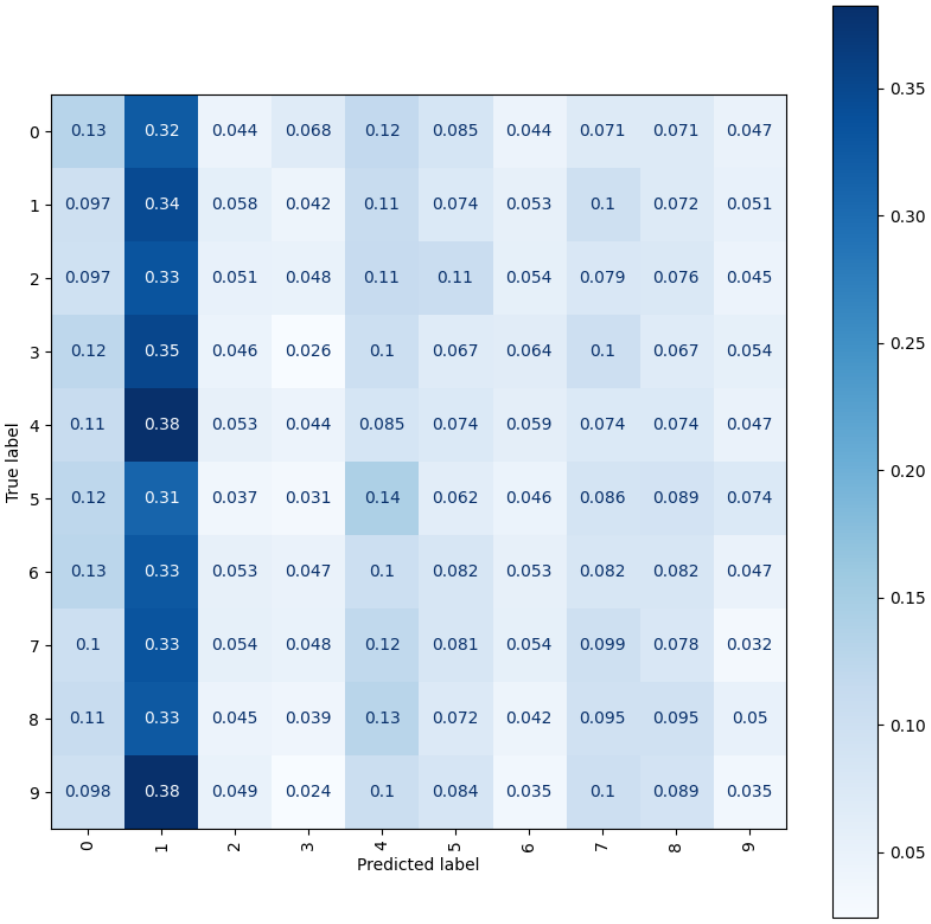
Obrázok 15 Confusion matica nad EEG pre 1. model 1. generátor

Následne som skúšal konfiguráciu prvého modelu a druhého generátora. Výsledkz su nasledujúce.

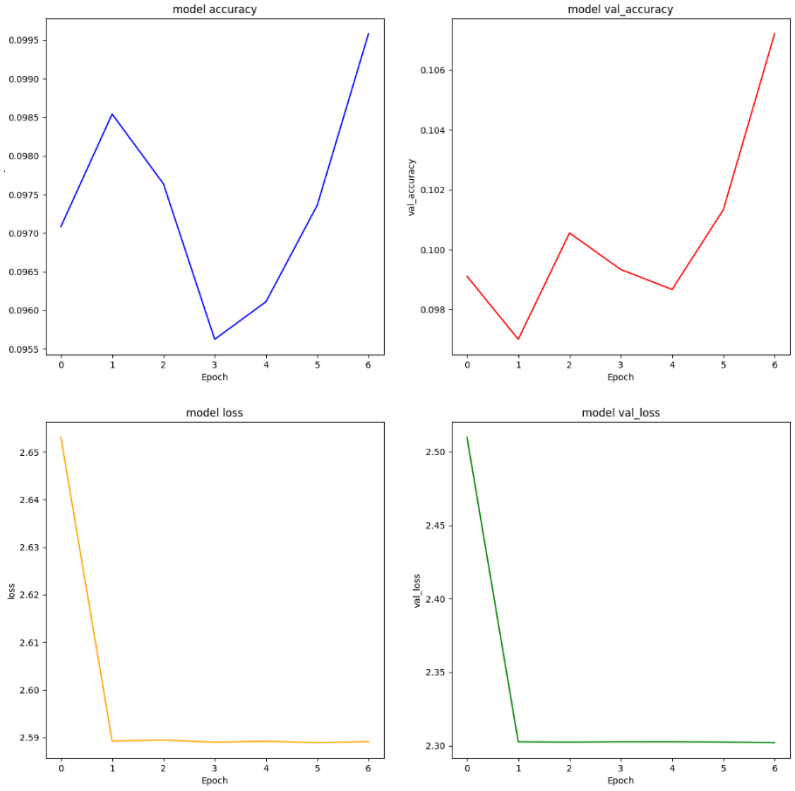


Obrázok 16 História trénovania 1. model 2. generátor

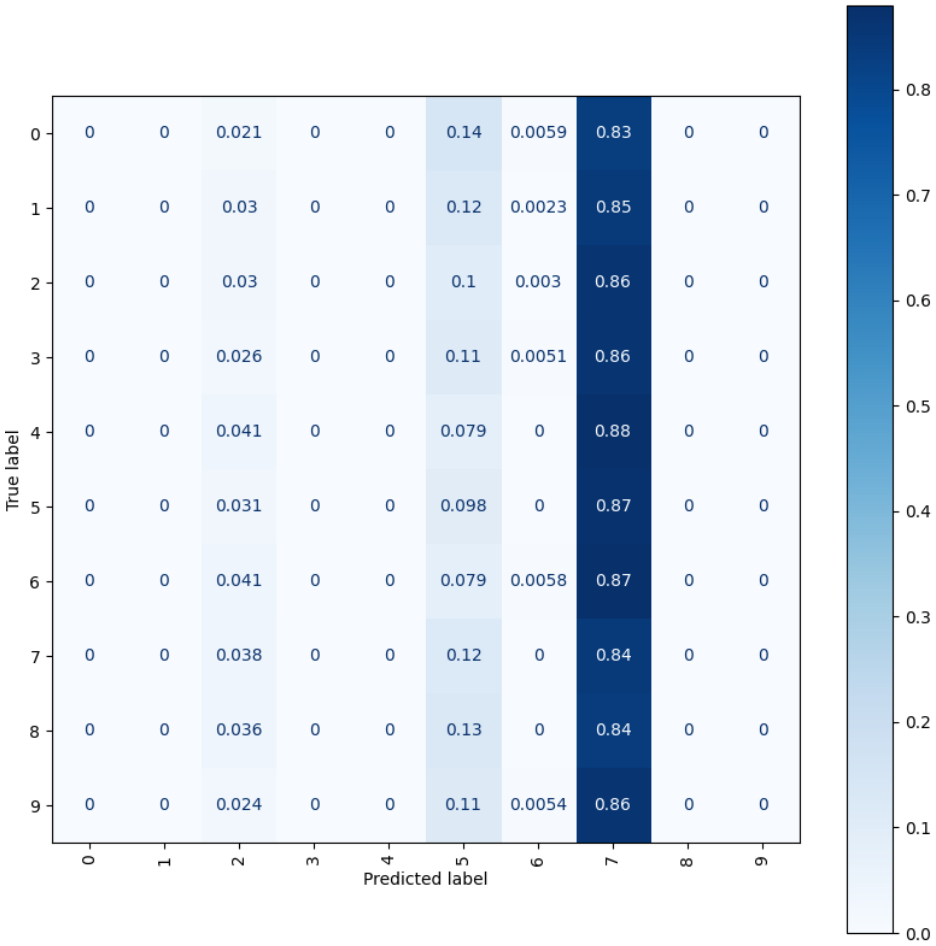
	precision	recall	f1-score	support
0	0.11	0.13	0.12	340
1	0.12	0.34	0.18	432
2	0.10	0.05	0.07	331
3	0.07	0.03	0.04	390
4	0.07	0.09	0.08	340
5	0.07	0.06	0.07	325
6	0.10	0.05	0.07	342
7	0.12	0.10	0.11	372
8	0.12	0.09	0.11	359
9	0.08	0.04	0.05	369
accuracy			0.10	3600
macro avg	0.09	0.10	0.09	3600
weighted avg	0.10	0.10	0.09	3600



Následne som trénoval RESNET-50 model. Výsledky and prvým generátorom sú nasledujúce.



	precision	recall	f1-score	support
0	0.00	0.00	0.00	340
1	0.00	0.00	0.00	432
2	0.09	0.03	0.04	331
3	0.00	0.00	0.00	390
4	0.00	0.00	0.00	340
5	0.08	0.10	0.09	325
6	0.20	0.01	0.01	342
7	0.10	0.84	0.18	372
8	0.00	0.00	0.00	359
9	0.00	0.00	0.00	369
accuracy			0.10	3600
macro avg	0.05	0.10	0.03	3600
weighted avg	0.04	0.10	0.03	3600



Záver:

Úlohou projektu bolo pripraviť CNN neurónové siete na klasifikáciu cifier z dát obrázkov cifier a EEG dát. Podarilo sa nám pripraviť model pre klasifikáciu nad obrázkami, kde sme mali dobré výsledky aj pri použití len 16 filtrov.

Model pre klasifikáciu and EEG dátami bol však problemový. Pripravili sme dva modely, kde jeden obsahoval tri konvolučné vrstvy, a druhý bol namodelovaný pomocou RESNET-50 hĺbkovej neurónovej siete. Ani v jednej z konfigurácii model nedosahoval dostačujúce výsledky presnosti. Loss funkcia padala veľmi pomaly, a presnosť týchto modelov bola pod 15 percent. Na trénovanie sme skúsili odstrániť udalosti, ktoré nepredstavovali cifru. Avšak ani na týchto dátach neurónová sieť nedosahovala lepšie výsledky.

Zistením tohto projektu je fakt, že na EEG dátach nie je ľahké nájsť podobné atribúty, ktoré by konvolučná neurónová sieť vedela detegovať pomocou filtrov.