Fakultet inženjerskih nauka
Univerziteta u Kragujevcu

Tema:

Konvoluciona neuronska mreža za prepoznavanje cifara

student:
Nikola Mitrevski
400/2021

predmetni profesor:
Vesna Ranković
predmetni asistent:
Tijana Šušteršič

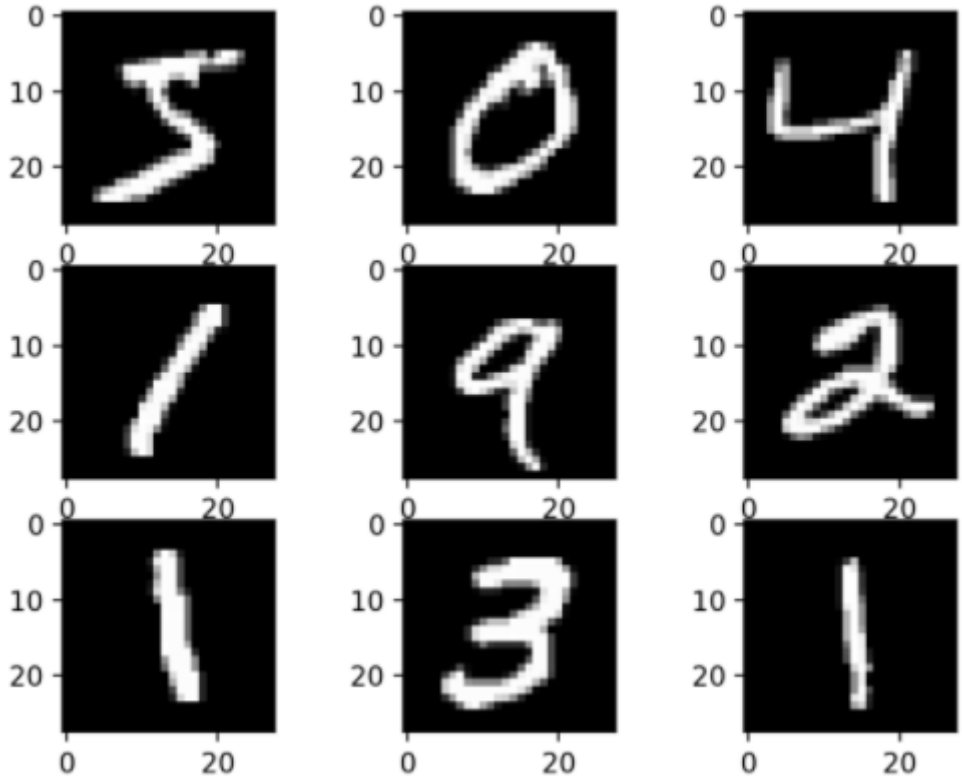
Kragujevac 2021.

Sadržaj:

1 Uvod

U ovom radu biće objašnjena konvoluciona neuronska mreža(KNM) za klasifikaciju cifara na osnovu MNIST skupa podataka.

MNIST(Modified National Institute of Standards and Technology) skup podata je skup podataka koji se sastoji od 60000 slika, predstavljene nijansama sive boje, veličine 28x28 piksela, gde svaka slika predstavlja jednocifren broj između 0 i 9.



Slika 1 Devet uzorka iz MNIST skupa podataka

Dakle zadatak je klasifikovati datu sliku u jednu od 10 klasa.

Razlog korišćenja KNM je taj što KNM postiže preko 99% tačnosti(stopa greške je između 0.2 i 0.4%).

2 Učitavanje MNIST skupa podataka

U kodu ispod je dat prikaz funkcije „load_dataset“ koja se koristi za učitavanje MNIST skup podataka i podelu istih na skup podataka za treniranje i na skup podataka za testiranje mreže.

```
# load train and test dataset
def load_dataset():
        # load dataset
        (trainX, trainY), (testX, testY) = mnist.load_data()
        # reshape dataset to have a single channel
        trainX = trainX.reshape((trainX.shape[0], 28, 28, 1))
        testX = testX.reshape((testX.shape[0], 28, 28, 1))
        # one hot encode target values
        trainY = to_categorical(trainY)
        testY = to_categorical(testY)
        return trainX, trainY, testX, testY
```

3 Skaliranje vrednosti svakog piksela

Vrednost svakog piksela za svaku sliku iz MNIST skupa podataka je neoznačeni celi broj iz opsega 0 do 255(0 za crnu i 255 za belu boju).

Svaki piksel je potrebno skalirati jer KNM brže konvergira na skupu od [0..1], nego na skupu [0..255].

U kodu ispod je dat prikaz funkcije „prep_pixels“ koja se koristi za skaliranje vrednosti svakog piksela, tako što prvo pretvori vrednost svakog piksela u tip float, a zatim svaki piksel deli maksimalnom vrednošću piksela(255).

```
def prep_pixels(train, test):
	# convert from integers to floats
	train_norm = train.astype('float32')
	test_norm = test.astype('float32')
	# normalize to range 0-1
	train_norm = train_norm / 255.0
	test_norm = test_norm / 255.0
	# return normalized images
	return train_norm, test_norm
```

4 Definisanje modela KNM

Model KNM se sastoji od dva glavna dela(prednji i zadnji).

Prednji deo se sastoji od konvolucijskog sloja i sloja objedinjavanja.

Konvolucijski sloj(Conv2D) je sastavljen od skupa filtera(32 filtera, svaki veličine 3x3) koji se mogu naučiti i koji služe za transformaciju svakog dela slike.

Sloj objedinjavanja(MaxPool2D) je filter(veličine 2x2) koji služi za smanjenje uzorkovanja(gleda 2 susedna piksela i bira maksimalnu vrednost).

.

Što je veća dimenzija objedinjavanja, to je manje uzorkovanje važnije.

Zadnji deo se koristi za predviđanje.

U kodu ispod je dat prikaz funkcije „define_model“ koja se koristi za definisanje modela KNM.

```
def define_model():
	model = Sequential()
	model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', input_shape=(28, 28, 1)))
	model.add(MaxPooling2D((2, 2)))
	model.add(Flatten())
	model.add(Dense(100, activation='relu', kernel_initializer='he_uniform'))
	model.add(Dense(10, activation='softmax'))
	# compile model
	opt = SGD(lr=0.01, momentum=0.9)
	model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
	return model
```

5 Procena modela KNM

Model KNM će biti procenjen korišćenjem petostruke unakrsne validacije.

Vrednost k = 5 je izabrana zbog kraćeg vremena rada.

Svaki od 5 skupa podataka za obuku će imati oko 12000 uzorka(oko 20% uzorka će imati svaki skup od ukupnog skupa podataka za obuku).

Skup podataka za obuku se meša pre nego što se podeli, a mešanje uzorka se izvodi svaki put.

Model KNM će biti obučen sa 10 epoha obuke sa podrazumevanom veličinom epohe od 32 uzorka.

Skup podataka za testiranje će se koristiti za procenu performansi modela KNM.

U kodu ispod je dat prikaz funkcije „evaluate_model“ koja se koristi za procenu modela KNM.

```
# evaluate a model using k-fold cross-validation
def evaluate_model(dataX, dataY, n_folds=5):
	scores, histories = list(), list()
	# prepare cross validation
	kfold = KFold(n_folds, shuffle=True, random_state=1)
	# enumerate splits
	for train_ix, test_ix in kfold.split(dataX):
		# define model
		model = define_model()
		# select rows for train and test
		trainX, trainY, testX, testY = dataX[train_ix], dataY[train_ix], dataX[test_ix], 
dataY[test_ix]
		# fit model
		history = model.fit(trainX, trainY, epochs=10, batch_size=32, 
validation_data=(testX, testY), verbose=0)
		# evaluate model
		_, acc = model.evaluate(testX, testY, verbose=0)
		print('> %.3f' % (acc * 100.0))
		# stores scores
		scores.append(acc)
		histories.append(history)
	return scores, histories
```

6 Predstavljanje rezultata modela KNM

Postoje dva ključna aspekta koja treba predstaviti:

1. dijagnostika ponašanja modela tokom njegove obuke;
2. procena performansi modela.

Dijagnostika uključuje kreiranje linijskog grafikona koji prikazuje performanse modela na trening i test skupu tokom svake unakrsne validacije.

Dijagnostika je dragocena za dobijanje ideje o tome da li je model preopterećen, nedovoljno ili se dobro uklapa u skup podataka.

U kodu ispod je dat prikaz funkcije „summarize_diagnostics“ koja se koristi za kreiranje dijagnostike ponašanja modela KNM tokom njegove obuke.

```
# plot diagnostic learning curves
def summarize_diagnostics(histories):
	for i in range(len(histories)):
		# plot loss
		pyplot.subplot(2, 1, 1)
		pyplot.title('Cross Entropy Loss')
		pyplot.plot(histories[i].history['loss'], color='blue', label='train')
		pyplot.plot(histories[i].history['val_loss'], color='orange', label='test')
		# plot accuracy
		pyplot.subplot(2, 1, 2)
		pyplot.title('Classification Accuracy')
		pyplot.plot(histories[i].history['accuracy'], color='blue', label='train')
		pyplot.plot(histories[i].history['val_accuracy'], color='orange', label='test')
	pyplot.show()
```

Funkcija „summarize_diagnostics“ kreira jednu figuru sa dva podgrafikona, jedan za grešku, a drugi za tačnost, gde plave linije će ukazivati na performanse modela na skupu podataka za obuku, a narandžaste linije će ukazivati na performanse modela na skupu podataka za testiranje.

Ocene tačnosti klasifikacije prikupljene tokom svake unakrsne validacije se mogu sumirati izračunavanjem srednje vrednosti i standardne devijacije.

U kodu ispod je dat prikaz funkcije „summarize_performance“ koja se koristi za kreiranje procene performansi modela.

```
# summarize model performance
def summarize_performance(scores):
	# print summary
	print('Accuracy: mean=%.3f std=%.3f, n=%d' % (mean(scores)*100, std(scores)*100,
len(scores)))
	# box and whisker plots of results
	pyplot.boxplot(scores)
	pyplot.show()
```

7 Pokretanje modela i prikaz rezultata

U kodu ispod je dat prikaz funkcije „run_test_harness“ koja se koristi za pokretanje modela.

```
# run the test harness for evaluating a model
def run_test_harness():
        # load dataset
        trainX, trainY, testX, testY = load_dataset()
        # prepare pixel data
        trainX, testX = prep_pixels(trainX, testX)
        # evaluate model
        scores, histories = evaluate_model(trainX, trainY)
        # learning curves
        summarize_diagnostics(histories)
        # summarize estimated performance
        summarize_performance(scores)
```

Pokretanje modela se vrši na sledeći na sledeći način:

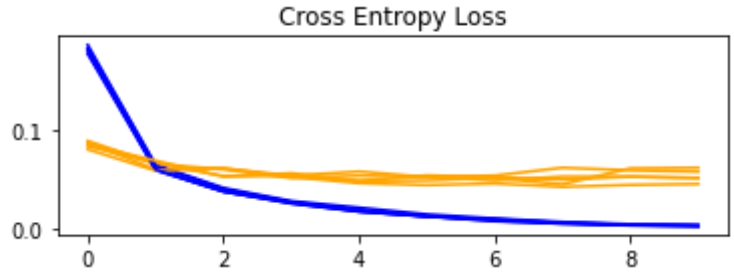
```
# entry point, run the test harness
run_test_harness()
```

Na slici 2 su prikazane tačnosti modela koje su dobijane prilikom njegove evaluacije.

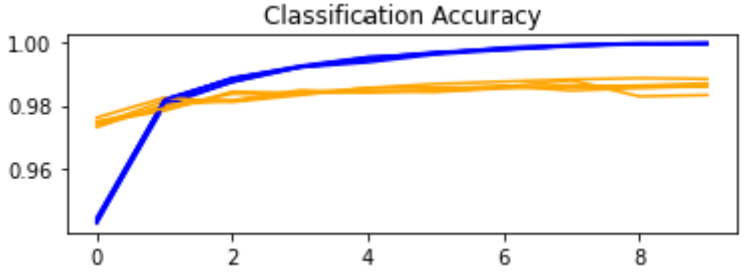
```
> 98.617
> 98.700
> 98.600
> 98.842
> 98.333
```

Slika 2 Tačnosti modela prilikom njegove evaluacije

Na slici 3 i 4 je dat grafički prikaz tačnosti i greške modela nakon njegove evaluacije, s kojih vidimo da nije došlo do prekomerenog prilagođavanja modela (overfitting), jer nema prevojne tačke(mesto gde funkcija prelazi iz konveksnosti u konkavnost i obrnuto).



Slika 3 Grafički prikaz greške modela nakon evaluacije



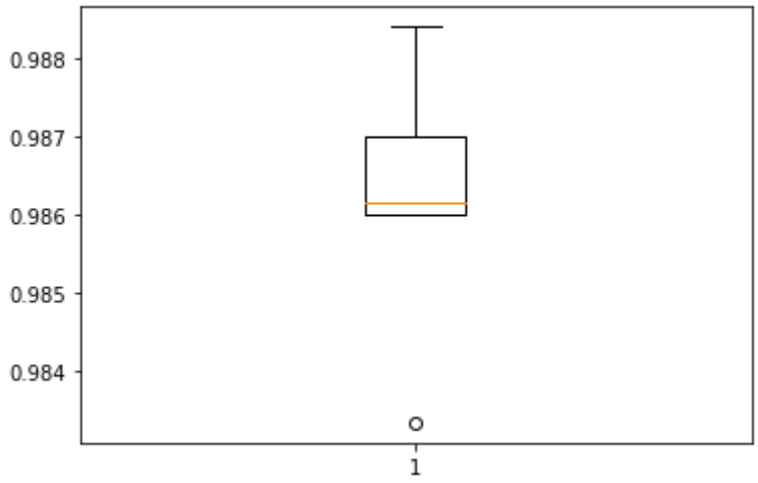
Slika 4 Grafički prikaz tačnosti modela nakon evaluacije

Na slici 5 je prikazano sumiranje performansi modela nakon njegove evaluacije.

```
Accuracy: mean=98.618 std=0.166, n=5
```

Slika 5 Sumiranje performansi modela

Na slici 5 je prikazan dijagram tačnosti modela nakon njegove evaluacije.



Slika 6 Dijagram tačnosti rezultata

8 Literatura

[1] Machine Learning Mastery -> How to Develop a CNN for MNIST Handwritten Digit Classification, link: https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-from-scratch-for-mnist-handwritten-digit-classification/, 06.11.2021(22:18)

[2] kaggle -> Introduction to CNN Keras - 0.997 (top 6%), link: Introduction to CNN Keras - 0.997 (top 6%) | Kaggle, 06.11.2021(22:21)