Crtanje i prepoznavanje 2D oblika korišćenjem neuronskih mreža i genetskih algoritama

Seminarski rad u okviru kursa
Računarska inteligencija
Matematički fakultet

Pejović Petar, Nikolić Igor
petar.pejovic8@gmail.com, igor.nikolic032@hotmail.com

15. juni 2019

Sažetak

U ovom radu biće predstavljen program za crtanje i prepoznavanje 2D oblika korišćenjem neuronskih mreža i genetskih algoritama. Na početku će biti opisan problem i baze koje su korišćene. Nakon toga, predstavljeno je rešenje za ovaj problem i dobijeni rezultati.

Sadržaj

1 Uvod

Program se bavi prepoznavanjem 2D oblika, odnosno prepoznavanje trouglova, krugova i kvadrata. Skup trening podataka koji se koriste za učenje neuronske mreže automatski je kreiran. Na slikama se nalaze isključivo tri prethodno navedena geometrijska oblika i ništa osim toga. Cilj je da se na osnovu nacrtanog oblika prepozna o kom geometrijskom telu se radi. Program je zasnovan na mašinskom učenju, gde je glavni zadatak je naći dobar klasifikacioni model. Sastoji od tri faze. U prvoj fazi se vrši automatsko generisanje slika koje će predstavljati ulaz. Nakon toga, generisane slike dele se u dva skupa, za trening i za test. Skup za trening je veći, oblično obuhvata 80 procenata ulaza, dok ostatak pripada test skupu. Učenje modela se vrši na trening podacima, da bi se na kraju taj model testirao na podacima za test. U implementaciji ovog programa korišćene su veštačke neuronske mreže i genetski algoritam.

2 Opis problema i baze

Projekat se bavi klasifikacijom geometrijskih oblika pomoću veštačkih neuronskih mreža i genetskog algoritma. Zadatak je da se na osnovu nacrtanog oblika, odredi kojoj klasi oblik pripada.

2.1 Opis baze

Baza podataka se sastoji iz slika za trening i test. U njima se nalaze tri foldera koji nazivaju Triangle, Circle i Rectangle u kojima su slike odgovarajućih objekata. Baza je generisana upotrebom jezika ObjectiveC kako bi se napravile najpogodnije slike za ulaz. Pored toga, dodate su i ručno nacrtane skice oblika, kako bi se povećala mogućnost pogađanja oblika nacrtanog u našem programu. Za svaki oblik pravi se 59 slika, od kojih je 50 automatski generisano i 9 ručno nacrtano. Primeri oblika iz baze nalaze se na slici 1.

Slika 1: Krug Kvadrat Trougao

3 Opis rešenja

Problem prepoznavanja oblika može se rešiti kombinacijom neuronskih mreža i genetskog algoritma. Program se sastoji iz četiri faze:

- Automatsko kreiranje slika
- Pretprocesiranje korišćenjem genetskog algoritma
- Učenje modela na trening podacima
- Testiranje modela

3.1 Automatsko kreiranje slika

U svakom folderu nalaze se slike odgovorajućih objekata. Primer koda za generisanje trouglova se nalazi na slici 1. Za kreiranje ostalih objekata koriste se slični algoritmi.

```
size = 100;
for i = 1:50
    R = randi([30 40]);
    x = randi([45 55]);
    y = randi([45 55]);

    %thetas = randi([1 12], [1, 3]);
    thetas = [0, pi/2, pi];
    p = round(x + R * cos(thetas));
    n = round(y + R * sin(thetas));

    figure('Position', [100, 100, 100, 100], 'Color', 'w', 'Visible
      ', 'off');
    fill(p, n, 'k')
    axis off
    f = getframe(gcf);

    imwrite(f.cdata, strcat('triangle-', num2str(i), '.png'))
    close all
end
```

Listing 1: Generisanje trouglova

3.2 Pretprocesiranje korišćenjem genetskog algoritma

Genetski algoritam jedan od vrsta algoritama Evolutivnog izračunavanja, gde se vrši evolucija nad linearnim genotipom. Na početku vrši se kreiranja početne populacije, koja je u našem programu reprezentovana pikselima slika. Nakon toga vrši se računanje prilagođenosti za svaku jedinku populacije. To se obavlja korišćenjem funkcije evoluate, gde se poredi vrednost jedinke i ciljne klase, a zatim i sortiranje jedinki po prilagođenosti. Kod se nalazi na slici 2.

```
    for gene in self.population:
      gene.evaluate(self.training_data, self.targets)

    self.population = sorted(self.population, key=lambda gene:
     gene.fitness)
    self.error = 1 - self.fittest().fitness
```

Listing 2: Funckija prilagođenosti

Nakon što smo sortirali gene na osnovu njihove prilagođenosti, biramo podskup najbolje prilagodjenih jedinki, veličine polovine populacije. Na tom podskupu, ruletskom selekcijom, vršimo izbor jedinki koje će ostaviti potomke. Kod ruletske selekcije se nalazi na slici 3.

```
choice = self.population[-self.popsize/2:]

    # Izracunavanje fitnes funkcije
    fitnesses = map(lambda x: x.fitness, choice)
    fitnesses /= np.sum(fitnesses)

    return np.random.choice(choice, n, p=fitnesses)
```

Listing 3: Ruletska selekcija

Nove jedinke se dobijaju dvopozicionim ukrštanjem. Ukrštanje se ponavlja sve dok se ne dostigne početna veličina populacije. Nakon ukrštanja, na novodobijene jedinke se primenjuje operator mutacije. Verovatnoća da će se dogoditi mutacija je 1 procenat.
Ovaj celokupan postupak se ponavlja sve dok ne bude zadovoljen kriterijum zaustavljanja. On će biti ispunjen nakon isteka unapred predviđenog broja generacija ili kada je nađeno prihvatljivo rešenje.

3.3 Učenje modela na trening podacima

Nakon faze pretprocesiranja vrši se učenje modela na trening podacima pomoću veštačke neuronske mreže (poglavlje 3.3.1). Za učenje neuronske mreže korišćen je Stohastički gradijentni spust. Ovaj algoritam se sastoji iz dve faze:

- Propagacija signala unapred
- Propagacija greske unazad

Za funkciju aktivacije korišćena je Sigmoidna funkcija i suma kvadratne greške uzeta je za funkciju cilja minimizacije. U ovom radu bavimo se klasifikacijom oblika, i iz tog razloga za aktivacionu funkciju izlaznog sloja je korišćena Softmax funkcija (poglavlje 3.3.4). Kod za učenje modela se nalazi na slici 4.

```
def gradient_descent(self, training_data, targets, epochs,
    test_data=None,
  vis=False):
    m = len(training_data)

    for i in range(epochs):
      nabla_b = [np.zeros(b.shape) for b in self.biases]
      nabla_w = [np.zeros(w.shape) for w in self.weights]
      self.errors.append(0) # Restartovanje greske
      self.train_accuracies.append(0)

      for tag, img in training_data:
        target = map(lambda x: int(x in tag), targets)
        delta_nabla_w, delta_nabla_b = self.backpropagate(img,
    target)


        for j in range(self.n - 1):
          nabla_w[j] += delta_nabla_w[j]
          nabla_b[j] += delta_nabla_b[j]

      # Azuriranje tezine i bajesa
      self.weights = [w-(self.learning_rate(i)/m)*nw for w, nw in
    zip(self.weights, nabla_w)]

      self.biases = [b-(self.learning_rate(i)/m)*nb for b, nb in
    zip(self.biases, nabla_b)]

      # Funkcija validacije neuronske mreze
      if test_data:
        self.test_accuracies.append(self.validate(targets,
    test_data))

      self.errors[-1] /= m # Normalizacija greske
      self.train_accuracies[-1] /= float(m)
```

Listing 4: Stohastički gradijentni spust

3.3.1 Neuronske mreže

Neuronske mreže predstavljaju jednu od paradigmi Računarske inteligencije. Sastoji se iz međusobno povezanih čvorova, koji se nazivaju

neuroni. Veštački neuroni napravljeni su po modelu biološkog neurona. Mreže imaju dva ili više slojeva, ulazni, izlazni i nula ili više skrivenih slojeva [2]. Vrste neuronske mreže su:

- jednoslojne
- višeslojne sa propagacijom unapred
- temporalne i rekurentne
- samoorganizujuće
- kombinovane mreže

3.3.2 Propagacija signala unapred

Propagacija signala unapred se sastoji u računanju vrednosti narednog sloja mreže. Kod za propagaciju unapred nalazi se na slici 5.

```
zs = []
    activations = [activation]
    z = activation

    # Iteriranje kroz skriveni sloj mreze
    for w, b in zip(self.weights[:-1], self.biases[:-1]):
      z = np.dot(w, activation) + b
      zs.append(z)
      activation = sigmoid(z)
      activations.append(activation)


    z = np.dot(self.weights[-1], activation) + self.biases[-1]
    zs.append(z)
    activations.append(softmax(z))
    return (activations, zs)
```

Listing 5: Propagacija signala unapred

3.3.3 Propagacija greške unazad

Kod propagacija greške unazad šalje se signal greške nazad ka ulaznom sloju, pri čemu se vrši izmena težinskih koeficijenata. Kod za propagaciju greške unazad nalazi se na slici 6.

```
def backpropagate(self, activation, target):
    # Inicijalizacija delta vrednosti
    nabla_b = [np.zeros(b.shape) for b in self.biases]
    nabla_w = [np.zeros(w.shape) for w in self.weights]

    activations, zs = self.feed_forward(activation)
    self.errors[-1] += square_error(target, activations[-1])
    if np.argmax(target) == np.argmax(activations[-1]):
      self.train_accuracies[-1] += 1

    # Izracunavanje izlazne vrednosti delte koriscenjem sigmoidne
     funkcije
    delta = softmax_prime(zs[-1]) * (activations[-1] - target)
    nabla_w[-1] = delta[:, None] * activations[-2][None, :]
    nabla_b[-1] = delta
    # Iteriranje kroz skriveni sloj mreze
    for i in xrange(2, self.n):
      # Izracunavnaje delte za svaki sloj
      delta = np.dot(self.weights[-i+1].T, delta) * sig_prime(zs[-
     i])
      nabla_w[-i] = delta[:, None] * activations[-i-1][None, :]
      nabla_b[-i] = delta

    return (nabla_w, nabla_b)
```

Listing 6: Propagacija greške unazad

3.3.4 Softmax

Softmax se uglavnom koristi kod problema klasifikacije podataka u N kategorija. Izlazi poslednjeg sloja se prosledjuju softmax funkciji, koja vraća vektor verovatnoća pripadnosti ulaza svakoj od klasa, odnosno vraća vektor y', gde su $y'_i = P(y = i|x)$, i = 1,...,n. Svi y'_i imaju vrednost između 0 i 1, tako da je suma svih y'_i jednaka 1.[1]

3.4 Testiranje modela

U poslednjoj fazi, vrši se testiranje dobijenog modela. Podaci za testiranje smešteni su u poseban folder. Za svaki oblik postoji 15 slika. Pored automatskog testiranja, postoji mogućnost provere jedne slike koja se može nacrtati u programu pozivanjem komande Draw. Ostale informacije o ovoj funkciji biće objašnjene u poglavlju 4. Pored računanja preciznosti, prilikom testiranja je korišćena matrica konfuzije. Kod za automatsko testiranje nalazi se na slici 7.

```
1000 def validate(self, targets, test_data):
     accuracy = 0.0
1002 matrica=np.zeros( (3, 3) )

1004 for tag, img in test_data:
       target = map(lambda x: int(x in tag), targets)
1006   activations, zs = self.feed_forward(img)
                         matrica[np.argmax(target)][np.argmax(
     activations[-1])]+=1
1008
       if np.argmax(target) == np.argmax(activations[-1]):
1010     accuracy += 1

1012     prikaziMatricuKonfuzije(matrica)
     return accuracy/len(test_data)
```

Listing 7: Testiranje modela

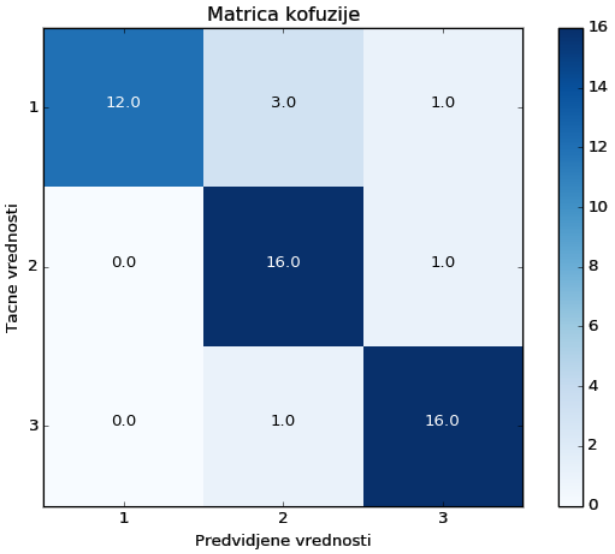
4 Crtanje i klasifikacija oblika

Pored automatskog kreiranja slika, omogućeno je i ručno crtanje oblika. Aplikacija za crtanje poziva se komandom draw (slika 2). Korisnik ima mogućnost da pomoću kursora nacrta oblik koji želi da program prepozna. Nakon toga, pozivom funkcije predict, korisnik dobija informaciju u koju klasu je njegova slika raspoređena i sa kojom verovatnoćom.

Slika 2: Aplikacija za crtanje

5 Eksperimentalni rezultati

Testiranjem modela, dobijeni su različiti rezultati. Rezultati zavise od izbora baze podataka. Eksperimentom smo zaključili da se dobija veća preciznosti ukoliko su objekti ispunjeni, odnosno, nije nacrtana samo njihova kontura. Nakon primene algoritma sa velikim brojem epoha, dobijena je preciznost od 90% na test skupu. Matrica konfuzije se nalazi na slici 3.



Slika 3: Matrica konfuzije

Promenom broja epoha kod genetskog algoritma i neuronske mreže dobijeni su različiti rezultati. Najbolji rezultati se ostvaruju za 1000 epoha za genetski algoritam i 1000 za neuronsku mrežu. Povećanjem parametara dobijaju se približno isti rezultati, dok smanjenjem rezultati opadaju.

5.1 Poređenje sa drugim radovima

Ne postoji veliki broj dostupnih radova na ovu temu. Tako da smo nas rad poredili sa radom koji prepoznaje ručno napisane i automatski generisane slike cifara, jer se radi o problemu kome se pristupa na sličan način [3]. Za prepoznavanje cifara korišćena je neuronska mreža sa propagacijom greške unazad. Neuronska mreža učena je sa 7500 ručno napisanih cifara i 2500 ištampanih. Nakon 30 epoha greška je bila samo 1.1%. Za test skup uzeto je 2000 ručno napisanih cifara i 700 ištampanih, i dobijena je preciznost klasifikacije od 96%. Ovaj algoritam se veoma dobro pokazao za prepoznavanje objekata (cifara) na slikama i iz tog razloga, u našem radu korišćena je neuronska mreža sa propagacijom signala u nazad.

6 Zaključak

Neuronske mreže u kombinaciji sa genetskim algoritmom problem prepoznavanja oblika rešavaju sa velikim uspehom. U ovom radu dobijeni su odlični rezultati, u velikom procentu program tačno klasifikuje oblike. Program se može dodatno unaprediti da radi i na ostalim 2D oblicima. Korišćenjem neuronskih mreža i genetskih algoritama mogu se rešiti i drugi problemi prepoznavanja sa velikom tačnošću.

Literatura

[1] Dive into deep learning. on-line at: https://www.d2l.ai/chapter_linear-networks/softmax-regression.html.

[2] Tehnički fakultet u Boru Dejan I. Tanikić, Univerzitet u Beogradu. Veštačke neuronske mreže, fazi logika i genetski algoritmi, 2016.

[3] J. S. Denker D. Henderson R. E. Howard W. Hubbard Y. Le Cun, B. Boser and L. D. Jackel. Handwritten Digit Recognition with a Back-Propagation Network, 1990.