DOMAĆI ZADATAK 2 – PREPOZNAVANJE CIFARA

Potrebno je implementirati algoritam za prepoznavanje rukom napisanih cifara koristeći klasifikator SVM. Algoritam kao obeležje koristi histogram orijentacije gradijenata (HOG). Računanje HOG-a se vrši na sledeći način::

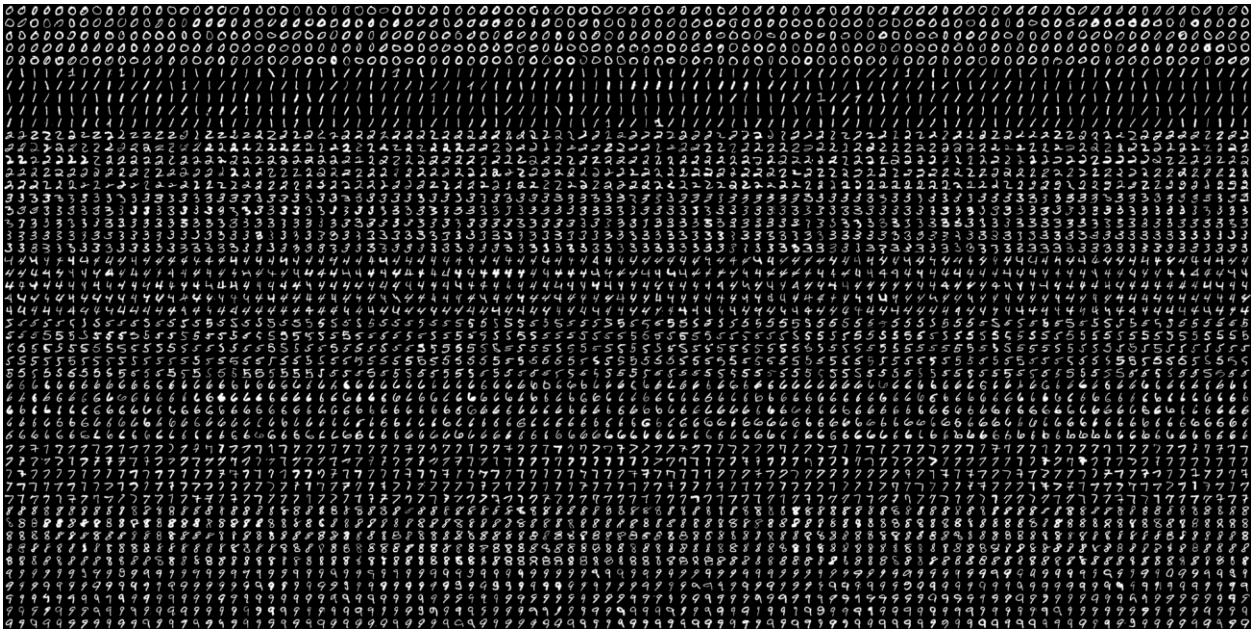
1. Konverzija slike iz RGB i YUV420 domen (ostatak algoritma ne uzima u obzir boju, radi samo nad Y komponentom).
2. Računanje horizontalnog i vertikalnog gradijenta primenom Sobel operatora nad *Y* komponentom slike (G_h i G_v).
3. Računanje orijentacije gradijenta za svaki piksel: $\theta = \arctan\left(\frac{G_y}{G_x}\right)$.
4. Računanje intenziteta gradijenta za svaki piksel: $G = \sqrt{{G_x}^2 + {G_y}^2}$
5. Podeliti sliku na blokove veličine 8x8 piksela.
6. Za svaki blok Izračunati histogram orijentacija gradijenata na sledeći način:
    a. Izračunati gornje granice podopsega za histogram: $gr_i = \frac{2*\pi*(i+1)}{histN}$, gde gr_i predstavlja gornju granicu opsega, *i* redni broj opsega, a *histN* ukupan broj podopsega (dužinu histograma). Granice su iste za sve blokove.
    b. Za svaku tačku iz bloka proveriti kom opsegu uglova pripada njena orijentacija θ. Potom odgovarajuću vrednost u histogramu uvećati za vrednost inzenziteta gradijenta G koji odgovara datoj tački.
    c. Nakon računanja histograma, sve vrednosti u histograu podeliti sa brojem piksela u bloku (64).
7. Konačan vektor obeležja formira se nadovezivanjem izračunatih histograma za sve blokove.

Izračunati vektori obeležja za svaku sliku predstavljaju ulazne podatke za SVM. Spram njjih potrebno je obučiti SVM klasifikator za prepoznavanje cifara. Uspešnost ispitivanja potrebno je izvršiti upotrebom krosvalidacije.

Opis baze podataka

Priložena baza podataka sadrži slike sa ručno napisanim ciframa belom bojom na crnoj pozadini. Sve slike sadrže tačno jednu cifru, koja je poravnata na sredinu slike. Sve slike su veličine 24*24 piskela. Baza sadrži ukupno 5000 slika, odnosno po 500 slika za svaku cifru [0-9]. Prikaz čitave baze dat je na slici 1. Pored baze podataka, dodatnih 20 slika (po 2 za svaku cifru) nalazi se u listi ulaznih slika za projekat ImageDSP. Ove

slike nisu deo baze, niti su obeležene laelama. One služe za proveru rešenja na novim slučajevima nakon uspešne realizacije.



Slika 1 - Baza podataka sa ručno napisanim ciframa

Opis priloženog koda

U okviru projekta ImageDSP nalazi se nezavršena implementacija algoritma za klasifikaciju. Potrebno je nad učitanom slikom (*inImgs*) primeniti algoritam, i potom u izlaznu sliku upisati o kojoj cifri se radi.

Projekat *TrainSVM* predstavlja pomoćni projekat (konzolnu aplikaciju) koji služi za obučavanje i validaciju SVM klasifikatora. Za potrebe klasifikacije korišćena je implementacija SVM pod nazivom LIBSVM. LIBSVM predstavlja javno dostupnu biblioteku (više informacija na https://www.csie.ntu.edu.tw/~cjlin/libsvm/). Kod biblioteke nalazi se u direktorijumu inc\svm i src\svm. Korišćene funkcije će biti opisane u daljem tekstu.

Program *TrainSVM* kao parametar prima željeni režim rada:

0	Obučavanje SVM
1	Predikcija na osnovu postojećeg modela
2	Krosvalidacija

U sva tri slučaja obrada se vrši nad ulaznom bazom podataka. Baza slika nalazi se u direktorijumu img_database. Sastoji se iz slika i datoteke koja sadrži listu slika i labela koje označavaju kojoj klasi slika priprada (koja cifra je u pitanju). Putanju do baze podataka i naziv datoteke sa listom slika i labelama moguće je promeniti u datoteci *SVMConsoleMain.cpp*, promenom promenljivih `dataBasePath` i `dataBaseName`.

U *main* funkciji vrši se poziv 4 funkcije. Prva funkcija služi za učitavanje baze podataka:

```
1. bool readDatabase(ImgDataBase& dataBase,string databasePath,string
   dataBaseName);
```

Funkcija učitava bazu podataka koja se nalazi na putanji *databasePath*, za koju se lista slika i labela nalazi u datoteci sa nazivom *dataBaseName*. Funkcija popunjava listu *database* i vraća informaciju o uspešnosti učitavanja.

Baza podataka predstavlja listu slika predstavljenih strukturom DBImage. Struktura DBImage data je sa:

`QImage* image`	Slika u RGB formatu predstavljena QT klasom Qimage
`int labelNumber;`	Redni broj klase kojoj slika pripada. Koristi se u fazi treniranja i za proveru uspešnosti klasifikacije u fazi prepoznavanja
`std::string label`	Tekstualni opis klase.

```
2. bool trainSVM(const ImgDataBase& dataBase, std::string SVMModelFileName);
```

Parametri funkcije su:

- `dataBase` – Baza slika nad kojom je potrebno uraditi obuku
- `SVMModelFileName` – Naziv pod kojim će biti sačuvan model SVM nakon obučavanja.

Funkcija *TrainSVM* u prvoj fazi priprema ulaz za SVM u formi strukture ***svm_problem*** koju očekuje LIBSVM. Klasa *svm_problem* sadrži tri polja: broj slika (*l*), pripadajuće klase za svaku sliku (*y*, niz džine *l*) i vektor obeležja za svaku sliku (*x*). Vektor obeležja je predstavljen skupom uređenih parova (indeks, vrednost) koji sadrži samo obeležja čija je vrednost različita od 0. Kraj vektora ozbačeb je elementom sa indeksom -1.

Parametri SVM su dati u fromi strukture ***svm_param***. Funkcija koja vrši postavljanje parametara na podrazumevane vrednosti naziva se *setDefaultParams* i u njoj je moguće praviti izmene parametara.

Računanje vektora obeležja za svaku sliku u bazi vrši se koristeći funkciju *calculateFeatureVector*. Vektor obeležja se potom zajedno sa rednim brojem klase dodaje u strukturu *svmProblemSet*.

Kada je problem upsešno specifiran neophodno je pozvati funkciju za obuku SVM, ***svm_train***. Ukoliko je treniranje uspešno vrši se čuvanje dobijenog modela u tekstualni fajl upotrebom funkcije *svm_save_model*.

Deklaracije funkcije *svm_train* je data sa:

- `svm_model *svm_train(const svm_problem *prob, const svm_parameter *param);`

gde *prob* predstavlja zadati problem, a *param* skup parametara za SVM: Povratna vrednost je obučeni model za klasifikaciju

```
3. bool crossValidateKFold(const ImgDataBase& dataBase, int K);
```

Parametri funkcije su isti kao i kod prethodne funkcije, sa izuzetkom parametra K koji predstavlja broj skupova na koji će ulazna baza podataka biti podeljena prlikom krosvalidacije. Priprema ulaznog problema se vrši na isti način kao i kod funkcije za obučavanje SVM. Nakon toga vrši se poziv funkcije za krosvalidaciju.

Krosvalidacija podrazumeva ispitivanje preciznosti klasifikatora za zadate parametre i ulazna obeležja. Krosvalidacija se vrši tako što se ulazni skup podataka nasumično podeli na K podskupova. Potom se za svaki od podskupova vrši iznova obučavanje klasifikatora sa preostalih K-1 skupova, dok zadati podskup predstavlja testni skup nad kojim se radi predikcija. Ukupno se dešava K ciklusa koji sadrže obuku i predikciju. Na kraju, računa se procenat uspešno pogođenih klasa.

Za krosvalidaciju korišćena je funkcija:

- `void svm_cross_validation(const struct svm_problem *prob, const struct svm_parameter *param, int nr_fold, double *target);`

Funkcija kao parametre prima problem i parametre za SVM, zatim broj skupova na koji se deli ulazni skup i niz čija je dužina jednaka broju slika u bazi podataka, u koji će biti smešteni rezultati predikcije.

4. `bool predictSVM(const ImgDataBase& dataBase, std::string SVMModelFileName);`

Funkcija služi za klasifikaciju koristeći unapred pripremljen model za SVM. Prvi korak u funkciji jeste učitavanje modela za SVM. Potom se za svaku sliku vrši poziv pomoćne funkcije *predictSingleImage* i računa broj uspešno klasifikovanih slika.

Funkcija:

- `int predictSingleImage(uchar input[], int xSize, int ySize, svm_model* svmModel)`

Za ulaznu sliku računa vektor obeležja. Nakon toga potrebno je pozvati funkciju ***svm_predict*** i proslediti joj SVM model i pripremeljeni ulazni vektor obeležja.

Deklaracije funkcije svm_predict je data sa:

- `double svm_predict(const struct svm_model *model, const struct svm_node *x);`

gde model predstavlja unapred obučeni SVM model, x vector ulaznih obeležja za zadatu sliku. Povratna vrednost funkcije je redni broj klase kojoj slika pripada.

ZADATAK

NAPOMENA: Sve slike su veličine 24*24 piksela, i dozvoljeno je to podrazumevati u Vašem kodu ukoliko će Vam olakšati implementaciju pojedinih delova. To znači da će slike uvek biti podeljene na 9 blokova, i da će dužina vektora obeležja biti 9*dužina_histograma. Tokom izrade možete koristiti projekat ImageDSP da prikažete rezultate međuobrade ukoliko će Vam to olakšati debagovanje (rezultat detekcije ivica, računanja orijentacije i slično).

Pokušajte da zaokružite rešenje kako ne biste gubili bodove za ispitivanje ispravnosti. Ukoliko ne uspete da realizujete čitav opisani algoritam iz zadatka 1, možete na osnovu onoga što ste dobro uradili da napravite svoj vektor obeležja, i na taj način realizujete zadatak 4. Npr. Histogram gradijenta po jednoj i drugoj osi razdvojeno, histogram intenziteta bez orijentacije, histogram orijentacija bez intenziteta, jedan histogram za čitavu sliku (bez podele na blokove) i slično.

1.1 Zadatak 1

Realizovati funkciju:

- `vector<double> calculateFeatureVector(const uchar input[], int xSize, int ySize)`

Funkcija vrši računanje vektora obeležja za prosleđenu sliku u RGB formatu i prosleđene dimenzije.

Parametri funkcije su:

- `input` – ulazna slika u RGB formatu
- `xSize` – horizontalna dimenzija slike
- `ySize` – vertikalna dimenzija slike

Izračunati vektor obeležja koristeći algoritam HOG opisan na početku ovog dokumenta za veličinu histograma (broj podopsega uglova) 9.

- Voditi računa da gradijent može biti i negativan, te rezultat filtriranja Sobel operatorima čuvati kao označene brojeve (možete koristiti i tipove sa pokretnim zarezom).
- Za računanje *arctan* možete iskoristiti funkciju *atan2* iz zaglavlja *cmath.* Funkcija vraća vrednosti u opsegu [-π, π). Radi lakšeg računanja možete rezultat pomeriti u opseg [0, 2π) dodavanjem vrednosti π na dobijeni ugao nakon računanja.

1.2 Zadatak 2

Za realizovanu funkciju za računanje obeležja uraditi krosvalidaciju i zabeležiti dobijeni rezultat u datoteku rezultat.txt. Za uspešno realizovan čitav algoritam očekivana je preciznost >90%.

Trenirati model nad čitavom bazom i sačuvati kao datoteku. Nakon toga isprobati rešenje u projektu *ImageDSP.* U rezultat.txt zabeležiti broj uspešnih predviđanja.

1.3 Zadatak 3

Pokušajte unaprediti rešenje promenom veličine bloka ili dužine histograma. Rezultat zabeležiti u datoteku rezultat.txt.

Takođe možete menjati parametre SVM (kernel koji se koristi, parametre za datu kernel funkciju).

1.4 Zadatak 4

U program dodati funkciju *CalculateFeatureVectorCustom*, koja izračunava vektor obeležja za sliku na način na koji Vi sami odaberete. Ponoviti sve navedeno u Zadatku 2 sa tim da će se umesto funkcije *CalculateFeatureVector* koristiti funkcija *CalculateFeatureVectorCustom.*

1.5 Zadatak 5

Napiasti dokumentaciju koristeći šablon dat u sklopu prethodnog ispitnog zadatka. U okviru dokumentacije je obavezno opisati predloženi algoritam iz zadatka 4, čak i ako i nije implementiran.

NAPOMENA: Kao rešenje okačiti datoteke sa izvornim kodom iz vašeg projekta, trenirani SVM model i dokumentaciju u .pdf formatu.