Klasifikacija slika upotrebom konvolutivne neuronske mreže

U ovom projektu je potrebno upotrebom konvolutivne neuronske mreže izvšiti klasifikaciju slika. Slike predstavljaju rendgenske snimke pluća. Svaka slika pripada određenoj klasi i potrebno je dizajnirati neuronsku mrežu koja će na osnovu zadatih slika naučiti da za svaku novu sliku odredi kojoj klasi ta slika pripada.

Postoje tri klase slika:

1. *Normal* – slike koje predstavljaju zdrava pluća
2. *Virus* – slike koje predstavljaju pluća zaražena virusom
3. *Bacteria* – slike koje predstavljaju pluća zaražena bakterijom

Dizajniranje ovakvih mreža predstavlja kompleksan posao i potrebno je voditi računa o velikom broju faktora kao što su npr. broj parametara konvolutivne neuronske mreže, broj slika za obučavanje, dimenzije slika, broj konvolutivnih slojeva, itd.

U nastavku će biti objašnjeni koraci pri izradi našeg rešenja, problemi sa kojima smo se suočili, parametri koje je bilo potrebno podesiti, itd.

Podaci

Kao što je prethodno rečeno, podatke za našu neuronsku mrežu čine slike rendgenskih snimaka pluća podeljene u tri klase. Veličina tog skupa je oko 5280 slika. Klasu *normal* čini oko 1340 slika, klasu *bacteria* oko 2530 slika, a klasu *virus* čini oko 1400 slika. Dizajnirajući prve verzije našeg rešenja, pokušavali smo kako sa velikim mrežama (oko 2 miliona parametara), tako i sa malim mrežama (oko 200 hiljada parametara). Problem koji smo imali jeste, da bez obzira na to koliko naša mreža kompleksna ili, pak, jednostavna bila, pri obučavanju mreže nismo mogli da pređemo 65-70% tačnosti mreže. Uočili smo da naši podaci nisu balansirani, tj. da imamo dosta više slika koje pripadaju klasi *bacteria* nego ostalim klasama. Iz tog razloga, odlučili smo da upotrebom *ImageDataGenerator*-a generišemo po još hiljadu slika za klasu *virus* i klasu *normal*. Nakon ovoga, uspeli smo da postignemo bolji rezultat.

Obzirom da je potrebno odvojiti određeni deo slika za validaciju i testiranje, odlučili smo se za sledeće - od ukupnog broja slika, 20% smo odvojili za validaciju, a 80% za obučavanje neuronske mreže. Za testiranje, nismo želeli da uzimamo slike iz početnog skupa podataka, iz dva razloga:

1. Uzimanjem slika iz početnog skupa, smanjujemo broj slika za obučavanje, te tako smanjujemo mogućnost da se naša mreža „susretne“ sa što više različitih slika.
2. Oko 2 hiljade slika koje smo generisali kako bismo izbalansirali broj slika za svaku klasu je generisano na osnovu postojećih slika tih klasa primenom zumiranja, rotiranja, ... Na ovaj način, smatramo da su te slike slične sa slikama iz obučavajućeg skupa, te da tačnost našeg sistema neće biti realna.

Istražujući, uspeli smo da nađemo bazu podataka sa preko 600 novih slika, takođe, podeljenih u tri klase. Svaka klasa poseduje oko 200 slika.

Pokušali smo generisanjem novih slika, uz pomoć *ImageDataGenerator*-a, da povećamo naš skup podataka, kako bismo poboljšali performanse naše mreže, međutim, to nije dalo značajne rezultate.

Dimenzija slike

U našem skupu podataka, različite slike imaju različite dimenzije (1560x1668, 2024x2036, 4248x3480, ...). Jedan od glavnih problema ovde jeste taj što neuronska mreža zahteva da sve slike budu iste dimenzije, tako da je potrebno sve slike skalirati na istu dimenziju. Drugi problem jeste što su dimenzije slika prevelike, te za ovako velike slike potreban je veliki broj konvolutivnih slojeva, a samim tim i veliki broj parametara. Svođenje slika na istu dimenziju je rešeno uporebom biblioteke *cv2* i funkcije *resize* iz te biblioteke koja smanjuje dimenziju slike na zadatu dimenziju. Dimenzija slike koju smo koristili na ulazu u neuronsku mrežu iznosi 64x64. Izučavajući literaturu, u problemima sličnim našem, koriste se dimenzije slika od 28x28 do 256x256, pa čak i 512x512. Pokušavali smo sa raznim dimenzijama slike, međutim, dimenzija od 64x64, u konačnosti, je dala najbolje rezultate.

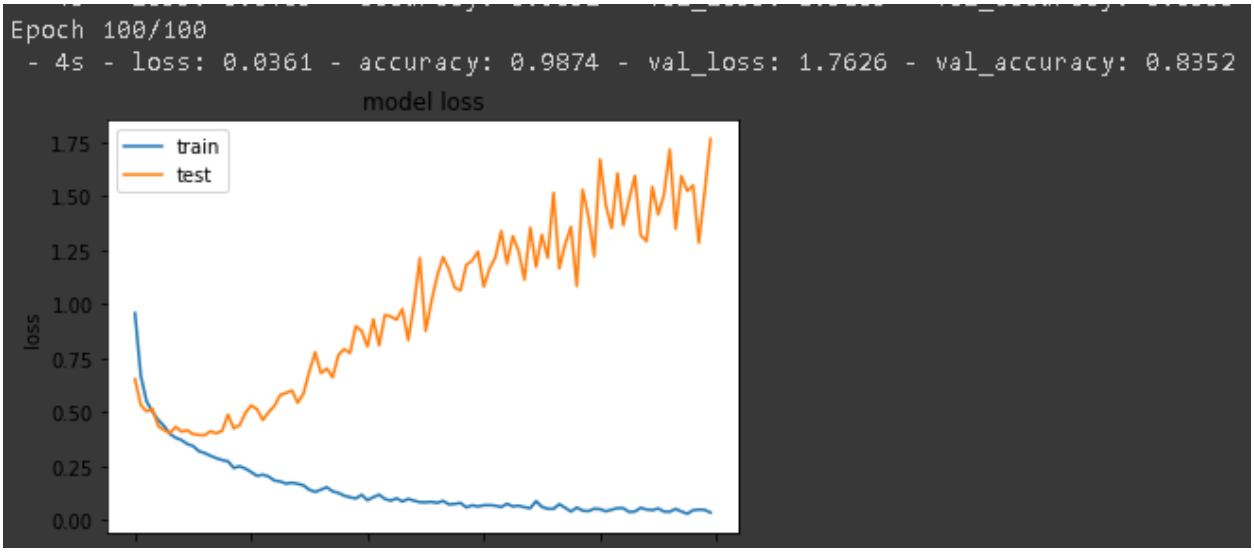
Takođe, pored same dimenzije slika, pojedine slike imaju samo jedan „sloj“, tj. one su crno bele, dok su druge RGB slike, tj. slike u boji. Zbog ovoga su sve slike prilikom učitavanja transformisane da budu crno bele, tj. dimenzije 64x64x1.

Arhitektura mreže

Dizajniranje arhitekture neuronske mreže zahteva podešavanje velikog broja parametara, definisanje konvolutivnih slojeva, regularizacije, itd. U nastavku će biti predstavljena naša neuronska mreža, kao i ponašanje same mreže usled različitih izmena mreže.

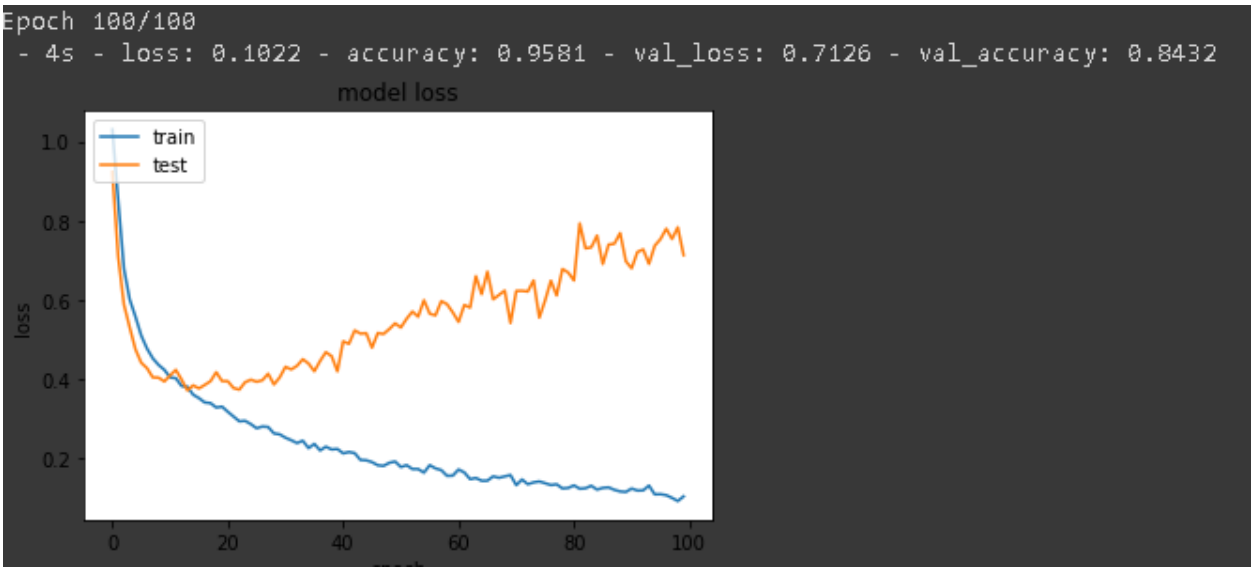
Naša krajnja konvolutivna neuronska mreža ima oko 128 hiljada parametara koje je potrebno obučiti. Takođe smo pokušavali i sa mrežama od 200 hiljada, 500 hiljada, milion, pa čak i mrežama sa 2-3 miliona parametara. Mreže sa jako velikim brojem parametara su se pokazale veoma neefikasnim i dale su veoma loše rezultate, pre svega jer je veličina našeg skupa podataka suviše mala da bi se mogli obučiti svi parametri te mreže. Mreže sa oko 500 hiljada parametara i manje su davale bolje rezultate u odnosu na veće mreže, međutim, vreme potrebno za njihovo obučavanje je bilo daleko veće nego vreme potrebno da se obuče mreže od 100 hiljada parametara, a rezultat je u određenim slučajevima bio isti ili lošiji od mreža sa 100 hiljada parametara.

Nakon što smo odredili da mreže od 100 hiljada do 200 hiljada parametara daju najbolje rezultate, krenuli smo sa analizom rezultata. Analizirajući rezultate obučavanja prikazane na grafiku jednog modela naše neuronske mreže (Slika 1), došli smo do zaključka da naša mreža ima *overfitting.* Naime, mreža postiže rezultate od 98% tačnosti sa greškom od 0.036 prilikom treniranja, dok su rezultati za validaciju 83% tačnosti i greška od 1.726. Ovo jasno ukazuje na to da mreža ima problem sa *overfitting*-om, što se jasno može videti i na grafiku na slici 1.

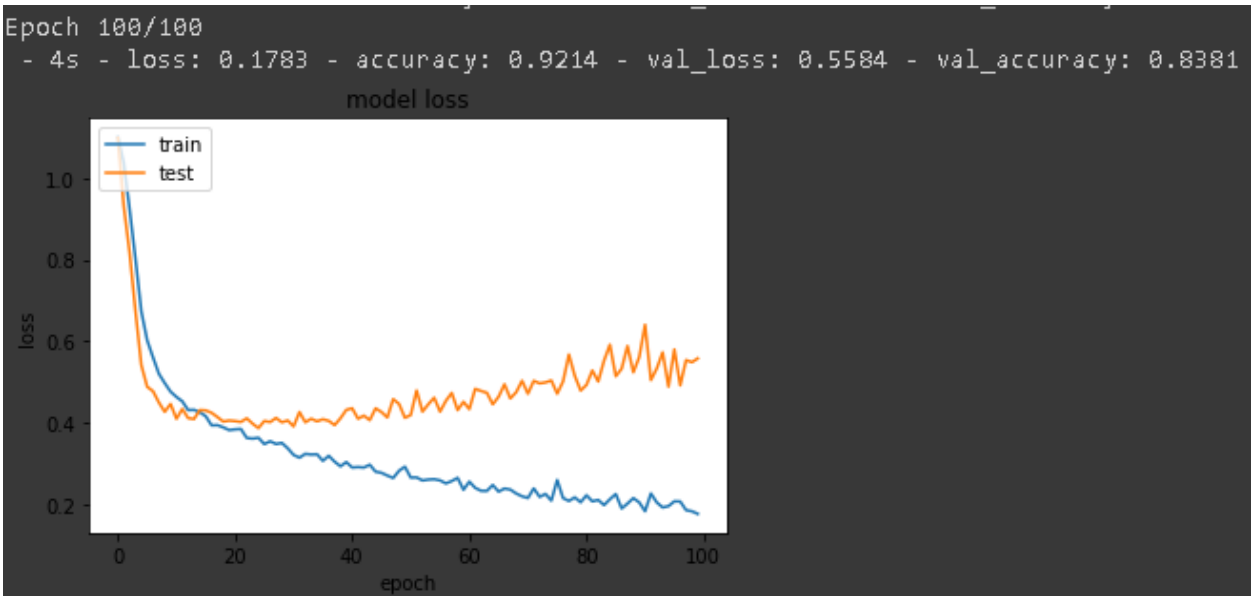


Slika 1

Da bismo rešili ovaj problem, odlučili smo da primenimo određene tehnike za rešavanje problema *overfitting*-a. Jedan od prvih načina za rešenje ovog problema jeste uvođenje *dropout*-a, koji je objašnjen na predavanju, te se stoga nećemo baviti s tim kako on radi. Postepenim dodavanjem se jasno može videti da povećanjem *dropout*-a, *overfitting* mreže se postepeno smanjuje. Ovo smanjenje je uočljivo i na grafiku (Slika 2, Slika 3).



Slika 2



Slika 3

Iz više različitih kombinacija sa *dropout*-om, došli smo do rešenja koje je pokazalo najbolje rezultate. Odlučili smo da *dropout* ubacimo između svakog sloja naše neuronske mreže , kako konvolutivnih, tako i običnih slojeva neuronske mreže. U prvim konvolutivnim slojevima, *dropout* iznosi oko 0.2, dok u kasnijim slojevima oko 0.3-0.4. *Dropout* između obična dva neuronska sloja iznosi 0.5. Pokušavali smo sa većim *dropout*-om, međutim, tada smo dobijali kontra efekat, mreža je postizala sve lošije rezultate. Razlog ovoga je sledeći - ako imamo veliki *dropout* u konvolutivnim slojevima, ti slojevi nisu u mogućnosti da izvuku bitne karakteristike sa slike jer se veliki broj parametara u tom slučaju „izbacuje“.

Pored primene *dropout*-a, problem *overfitting*-a, iako umanjen i dalje je ostao prisutan. Sledeći korak u rešavanju ovog problema jeste primena funkcija regularizacije. Postoje *l1* i *l2* funkcije regularizacije. U literaturi koju smo pronašli, *l2* regularizacija je preporučena za rešavanje problema *overfitting*-a, dok je *l1* regularizacija preporučena za rešavanje problema *underfitting*-a. Upotrebom *l2* regularizacije u kombinaciji sa *dropout*-om, uspeli smo u velikoj meri da rešimo problem *overfitting*-a. Posledica upotrebe ovih tehnika je daleko manja preciznost za vreme obučavanja mreže, ali i veća vrednost greške. Međutim, pored toga, rezultati koje smo dobili sa test skupom su daleko bolji nego pre rešavanja ovog problema. Razlog je taj što sada naša mreža ne uči „napamet“, nego uspeva da prepozna bitne karakteristike slike i da na osnovu toga donese zaključak kojoj klasi pripada slika koju do tada nikada nije „videla“.

Nakon primene tehnika za smanjenje *overfitting*-a, testiranja većih i manjih mreža, odlučili smo da naš model radi sa slikama dimenzija 64x64. Nakon svakog konvolutivnog sloja, dolazi m*ax pooling*, dimenzije 2x2, čime se dimenzija slike smanjuje dva puta nakon svakog konvolutivnog sloja. Tako da naša mreža ima 4 konvolutivna sloja, gde svaki sloj u sebi ima 64 sloja. Nakon toga dolaze dva obična potpuno povezana sloja neuronske mreže.

Svaki sloj konvolutivne neuronske mreže na izlazu ima funkciju aktivacije. Pokušavali smo sa *ReLu, Elu*, *SeLu* i *softmax* funkcijama aktivacije. Na kraju smo odlučili da koristimo *ReLu* funkciju aktivacije, za sve slojeve osim poslednjeg, koja je davala najbolje rezultate. U poslednjem sloju naše mreže odlučili smo da koristimo *softmax* funkciju aktivacije iz razloga što naš rezultat na kraju treba da odredi verovatnoću da određena slika pripada svakoj od tri klase. Upravo *softmax* funkcija aktivacije na izlazu daje vektor kategoričkih verovatnoća.

Obzirom da je zadatak projekta klasifikacija slika u tri klase, funkcija za računanje greške koja se koristi u našem modelu je *categorical crossentropy. Categorical crossentropy* funkcija greške se koristi za računanje greške kada imamo dve ili više klasa. Ova funkcija zahteva da se koristi takozvana *one hot* reprezentacija. To znači da ako imamo tri klase, neka slika npr. pripada prvoj klasi, ulaz ove funkcije predstavlja vektor [1, 0, 0], koji predstavlja da određena slika pripada prvoj klasi, a onda je potrebno proslediti i drugi vektor oblika [x, y, z], gde x, y i z predstavljaju redom verovatnoće, da ta slika pripada prvoj, drugoj ili trećoj klasi, koje je naš model izračunao upotrebom *softmax* funkcije aktivacije. Na osnovu ova dva ulaza, *categorical crossentropy* funkcija računa grešku našeg modela.

Na kraju je potrebno još odrediti optimizator našeg modela. Optimizator nam podešava gradijente, kako bismo postigli što tačnije rezultate. Postoje razni optimizatori, mi smo u našem modelu pokušali sa *SGD*, *Adam*, *Adamax*, *Adadelta*, *Adagrad*. *SGD* optimizator se pokazao kao najsporiji, tj., potrebno je dosta epoha da bismo došli do, za nas, zadovoljavajućeg rezultata. *Adadelta* i *Adagrad* su bili brži pri postizanju zadovoljavajućeg rezultata u odnosu na *SGD*, ali, ipak, njihov rezultat nije bio dobar, kao kod *Adam* i *Adamax* optimizatora. *Adam* i *Adamax* optimizatori su dali najbolje rezultate, tj., postigli su najveću tačnost za trening, validaciju, ali i za test. Razlika između ova dva optimizatora je u tome što *Adamax* optimizator radi malo sporije u odnosu na *Adam* optimizator. Potrebno je oko 20-30 epoha više da bi *Adamax* postigao najbolji rezultat. Prednost *Adamax* optimizatora jeste što je manje stohastičan u odnosu na *Adam*. Kod *Adam* optimizatora, tačnost modela varira u velikoj meri za više pokretanja, od 70% do 80%, dok kod *Adamax* modela, tačnost modela je uglavnom u intervalu od 76% do 80%. Takođe, u pojedinim situacijama pri upotrebi *Adam* optimizatora, znalo se dogoditi da model uopšte ne može da uči. Zbog ovih razloga, odlučili smo da iskoristimo *Adamax* optimizator, iako je u praksi u upotrebi više *Adam,* kod nas je pokazao malo lošije rezultate.

Konačan model i rezultati

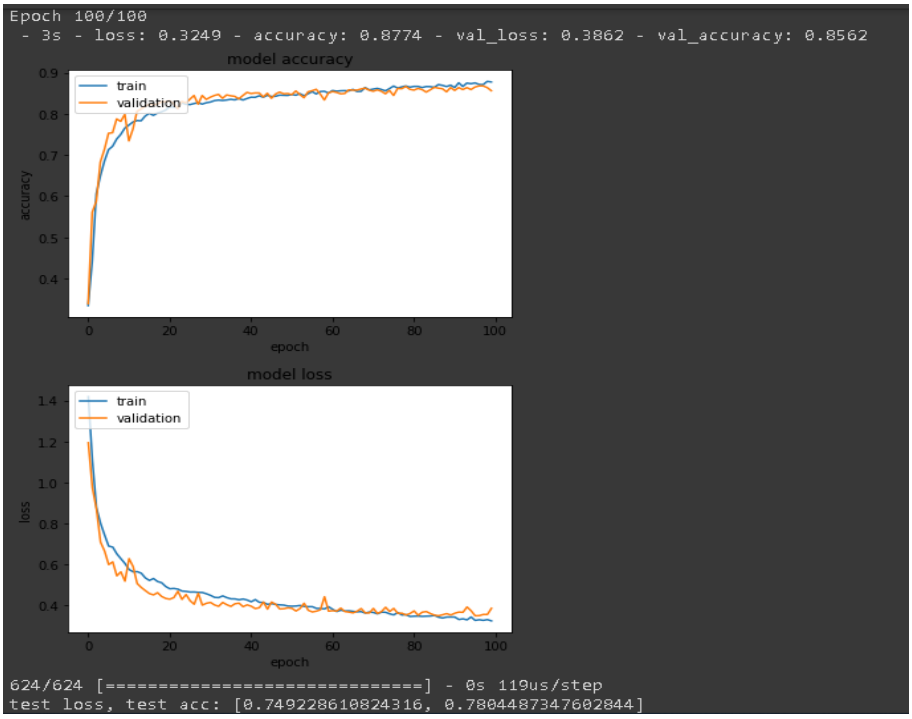
Rezultati, koje je naša konvolutivna neuronska mreža postizala, su varirali u velikoj meri u zavisnosti od arhitekture mreže, optimizatora, regularizacije, itd., o čemu je prethodno već bilo reči. Konačna verzija modela naše mreže može da se vidi na slici 4.

```
model.add(Conv2D(64, (3,3), input_shape = X.shape[1:]))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Dropout(0.2))

model.add(Conv2D(64, (3,3)))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Dropout(0.3))

model.add(Conv2D(64, (3,3)))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Dropout(0.3))

model.add(Conv2D(64, (3,3)))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Dropout(0.4))

model.add(Flatten())
model.add(Dense(64,activation='relu',kernel_regularizer=l2(regularizacija)))

model.add(Dropout(0.5))

model.add(Dense(3,kernel_regularizer=l2(regularizacija)))
model.add(Activation("softmax"))
```

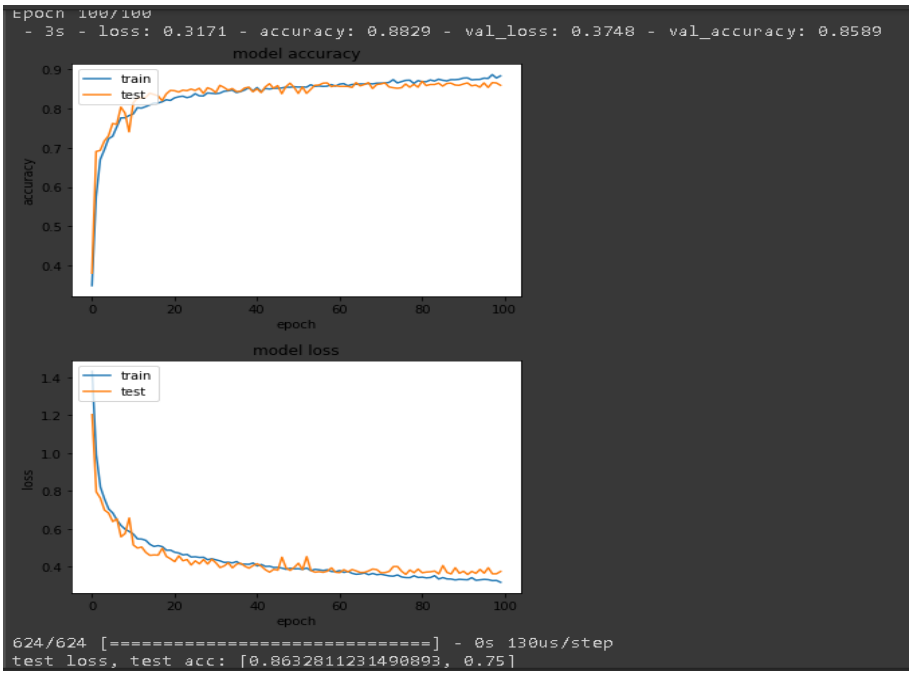
Slika 4

Model koristi, kao što je prethodno rečeno, *Adamax* optimizator. Broj epoha potreban da se obuči model kako bi postigao najbolji rezultat je 100. Ukupan broj slika koje se koriste za obučavanje modela je 7299. 80% ovih slika služi za trening, a 20% za validaciju, odnosno 5839 slika se koristi za trening, a 1460 slika za validaciju. Pored toga, za testiranje naše konvolutivne neuronske mreže koriste se još 624 slike, sa kojima se naša mreža do sada nije susrela. Tako da rezultat koji nam da test možemo posmatrati kao rezultat koji bi naša mreža postizala u praksi.

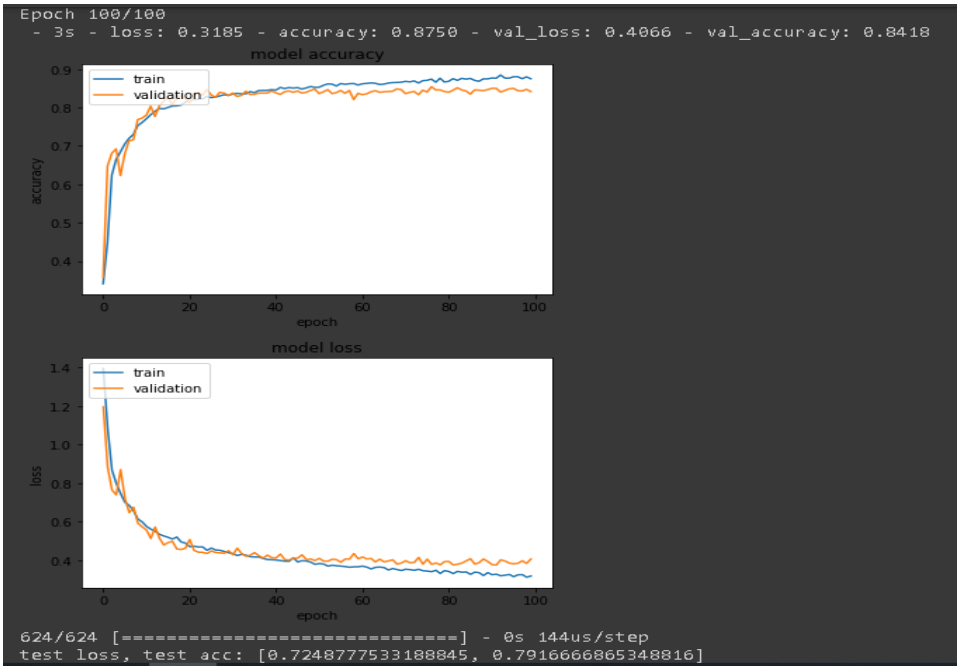
Rezultat koji se postiže je uglavnom oko 85%-88% za trening, što se vidi i na slici 5. Ova tačnost je išla daleko više, do čak 98%-99%, ali tada je model imao problem sa *overfitting*-om, te taj rezultat ne predstavlja stvarnu tačnost našeg model. Funkcija greške za trening iznosi od 0.31 do 0.35. Tačnost modela po pitanju validacije iznosi oko 84%-87%, na slici 5 možemo videti da tačnost i funkcija greške validacije u velikoj meri prate tačnost i funkciju greške tokom treninga, što nam pokazuje da model nema problema sa *overfitting*-om. Funkcija greške za validaciju se kreće u intervalu od 0.35 do 0.39. Što se tiče ponašanja modela u „praksi“, tj. nad test podacima, model postiže tačnost od 75% do 78%, dok funkcija greške iznosi od 0.7 do 0.9.



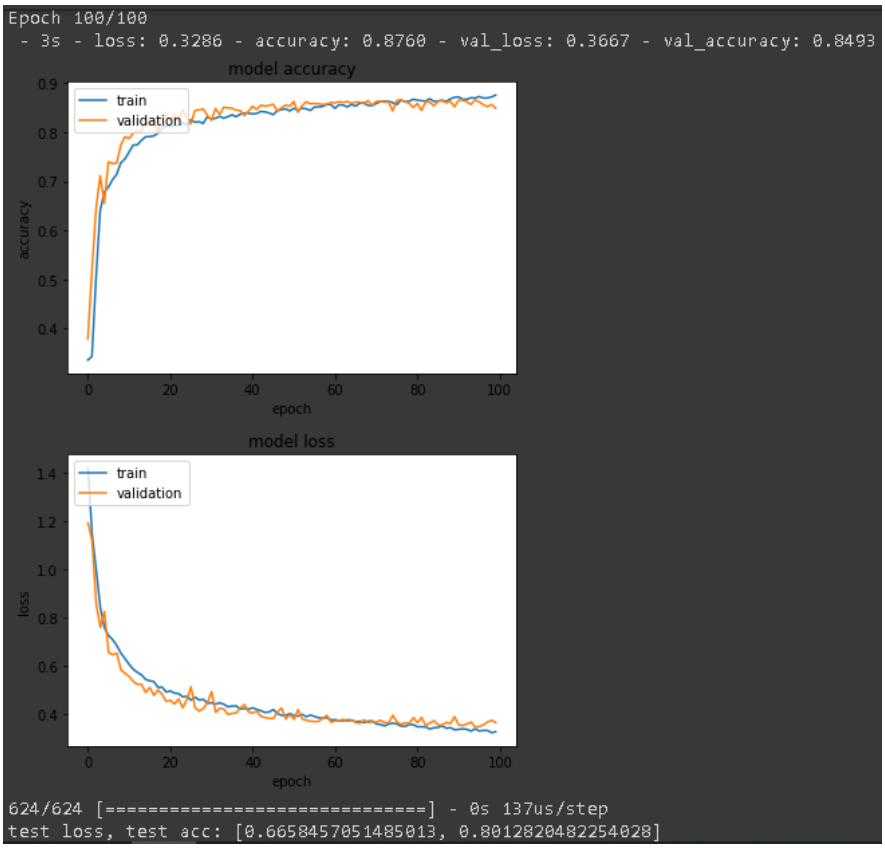
Slika 5



Slika 6



Slika 7



Slika 8