FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij

Računarstvo usluga i analiza podataka

SEMINARSKI RAD

Klasifikacija slika korištenjem konvolucijske neuronske mreže

Dražen Bertić i Davorin Miličević

Osijek, 2023. godine

Sadržaj

1. UVOD

Konvolucijske neuronske mreže (engl. CNN) su u posljednje vrijeme stekle veliku popularnost osobito kod strojnog učenja odnosno dubokog učenja gdje je fokus postavljen na klasifikaciju slika. Naime inicijalni rad koji je izradio Yann LeCun (LeNet) [1], u kojem je iznio temeljnu ideju klasifikacije znamenki (MNIST), te Alex Krizhevsky [1] koji je sa svojom AlexNet neuronskom mrežom prikazao dobre performanse kod detekcije i klasifikacije objekata sa slike koristeći CIFAR-10 set podataka, pokazalo se kao jedna pokretačka klica koja je iznjedrila, te opisala mogućnosti i velike prednosti konvolucijskih neuronskih mreža u algoritmima klasifikacije. Jedna glavna prednost konvolucijskih neuronskih mreža leži u tome da mreže imaju kapacitet unutar kojega mogu izdvojiti prikaze vizualnih podataka na višoj razini bez izdvajanja značajki (engl. feature extraction), što zahtijeva puno vremena, te je skup postupak kojim bi se stvorila domena na osnovu dobivenih značajki za strojno učenje. Konvolucijske neuronske mreže se sastoje od nekoliko filtera čiji se parametri mogu mijenjati kroz treniranje, odnosno prilagođavaju se na temlelju dobivenih podataka kroz slike[1]. Još jedna značajna prednost CNN-ova je njihova sposobnost da smanje broj parametara, te kao rezultat toga, računalno opterećenje biva manje dok se u međuvremenu postižu bolji rezultati. Također konvolucijske neuronske mreže optimiziraju konvolucijske jezgre (engl. kernels) čiji se zadatak ogleda u preslikavanju karakteristika slika na nižim dimenzionalnostima, omogućujući poboljšano upravljanje memorijom dok se zadržava dovoljan broj relevantnih informacija o slikama. Potpuno povezani slojevi (engl. FCN -> Fully Connected Layers) unutar CNN-ova omogućuju optimizirano mijenjanje konvolucijskih značajki i preslikavanje u diskretne kategorije (klase)[2].

Skup podataka CIFAR-10 uključuje 10 klasa, s 5 tisuća slika za treninga po klasi, te tisuću slika po klasi za testiranje [2].

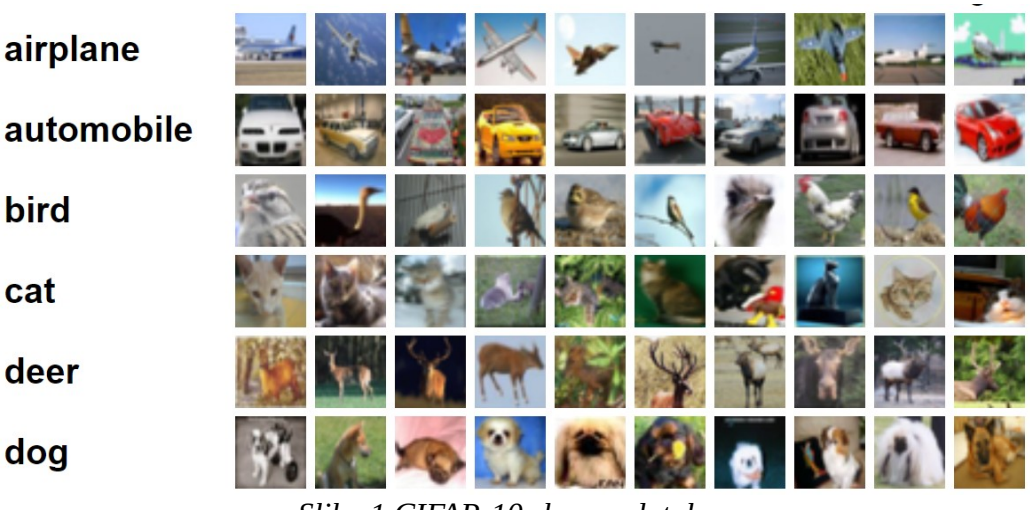
U ovom radu će biti prikazana metodologija izgradnje modela konvolucijske neuronske mreže, s definiranim filterima, arhitekturom i brzinom učenja, te rezultati testiranja modela, uz izradu prenosivog modela koristeći *TensorFlow Lite*, kako bi mogao biti implementiran na mobilnu aplikaciju.

2. METODOLOGIJA

U prvom koraku je potrebno izabrati adekvatne podatke nad kojima bi se izvršila klasifikacija objekata kroz slike. U ovom radu, izvor podataka je CIFAR-10 skup podataka, koji je odabran zbog same prilagodljivosti i pristupačne implementacije kroz biblioteke u Python.

2.1. Skup podataka

U ovom radu potrebno je klasificirati nasumično izvučene objekte kroz slike iz skupa podataka CIFAR-10. Skup podataka CIFAR-10 sastoji se od 60 000 jedinica (RGB) slika, gdje svaka slika ima dimenziju od 32 x 32 piksela. Skupove podataka je moguće svrstati u 10 razreda jednake veličine, gdje svaki razred (klasa) sadrži 6000 slika i razredi se uglavnom odnose na objekte poput (brodova, automobila, zrakoplova itd.) i životinja (mačke, psi, žabe itd.). Neki primjeri skupa podataka su ujedno s pripadajućim klasama prikazani na Slici 1. Ovakav skup podataka je podijeljen na skup za trening i testiranje. Skup za trening se sastoji od 50 000 slika, dok za test ima 10 000. Također klase nemaju podudaranja (npr. slike za automobil isključivo prikazuju automobile poput sedana i karavana bez terenskih vozila). Tako da nema poklapanja između automobila i kamiona[5].



Slika 1.CIFAR-10 skup podataka

CIFAR-10 skup podataka sadrži sljedeće klase:

- Zrakoplov
- Automobil
- Ptica

- Mačka
- Jelen
- Pas
- Žaba
- Konj
- Brod
- Kamion

Navedene klase su raspoređene kroz više kontejnera podataka, te svaka klasa je točno određena za svaku sliku istim redom kojim su klase nabrojane. Za svaki kontejner (engl. batch) su odabrane slike koje se neće poklapati prilikom podjele podataka na treniranje i testiranje. Primjerice test skup će sadržavati 1000 odabranih slika iz svih klasa podjednako.

Podatke je potrebno unutar okvira (*Juptyer Notebook)* učitati i to se radi pomoću ugrađenih funkcija koristeći biblioteku *Keras.* Učitavanje podataka je prikazano na Slici 2. gdje su podatci podijeljeni na *train* i *test* podskupove.

```
(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()

# Normalize pixel values to be between 0 and 1
train_images, test_images = train_images / 255.0, test_images / 255.0
```

Slika 2.Učitavanje podataka i normalizacija

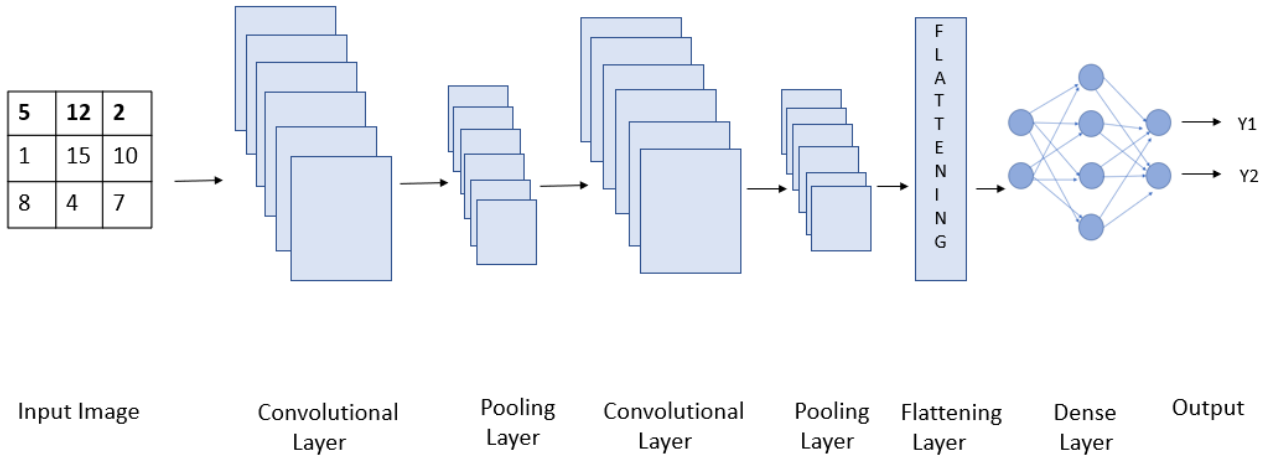
Ovakav način učitavanja podataka je moguć iz razloga što su slike unutar skupa podataka segmentirane (odnosno svaka slika sadržava jedan objekt, te svaka slika ima jednaku veličinu i 3 kanala). Prema Slici 2. se također vidi da je korištena normalizacija. Naime, proces normalizacije je bio potreban zbog skaliranja vrijednosti piksela unutar raspona od 0 do 1. Dakle, proces je rađen na temelju informacija o vrijednostima piksela unutar slika koji su se početno kretali od 0 do 255, te su predstavljali *unsigned integers* vrijednosti koje su se odnosile na prikaz boje slike. Nakon dijeljena s 255, dobiveni su normalizirani pikseli. Tim korakom je skup podataka bio spreman za rad s modelom [6].

2.2. Izgradnja modela

Za klasifikaciju slika potrebno je posegnuti za algoritmima iz dubokog učenja. To su u ovom slučaju konvulucijske neuronske mreže koje je potrebno dizajnirati na način u kojem bi dobili što bolje performanse i krajnje rezultate odluka. Stoga je bilo potrebno odrediti red i vrstu slojeva u arhitekturi mreže. Potom ključna konfiguracija su jezgre (kernels) i filteri, gdje je bitno postaviti veličinu matrice unutar slojeva. Naime, CNN se sastoji od ulaznog sloja, skrivenih slojeva u kojima se odvijaju ključne operacije. Unutarnji slojevi su slojevi (engl. Convolution & MaxPooling) unutar kojih se postavljaju određeni parametri. Potom slijedi izlazni FCN sloj, koji je zapravo prije toga ispeglan korištenjem *Flatten* sloja.

Zbog toga za konačne performanse je dobro mijenjati i testirati unutarnje slojeve konvolucije i udruživanja. No, prije toga je potrebno definirati veličinu kontejnera unutar kojih će se model trenirati, broj klasa i kroz koliko epoha je potrebno model da prođe.

Nakon toga je potrebno izgraditi model dio po dio, odnosno kroz definiranje slojeva i pripadajućih parametara. Definiranje slojeva je rađeno pomoću biblioteka *TensorFlow* i *Keras.* Model konvolucijske neuronske mreže je sljedeće arhitekture pojednostavljenog prikaza (Slika 3.):



Slika 3.Prikaz CNN korištene za klasifikaciju objekta na CIFAR-10 skupom podataka[7]

CONV->POOL->CONV->POOL->CONV->FLATTEN->DENSE->DENSE (32->64->64->1024->10).

U kodu bi to izgledalo na sljedeći način (Slika 4.):

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10))
```

Slika 4. Arhitektura modela izrađena koristeći Keras biblioteku

Detaljniji prikaz (Slika 5.) modela konvolucijske neuronske mreže se može izvesti koristeći funkciju *model.summary().*

```
model.summary()
```

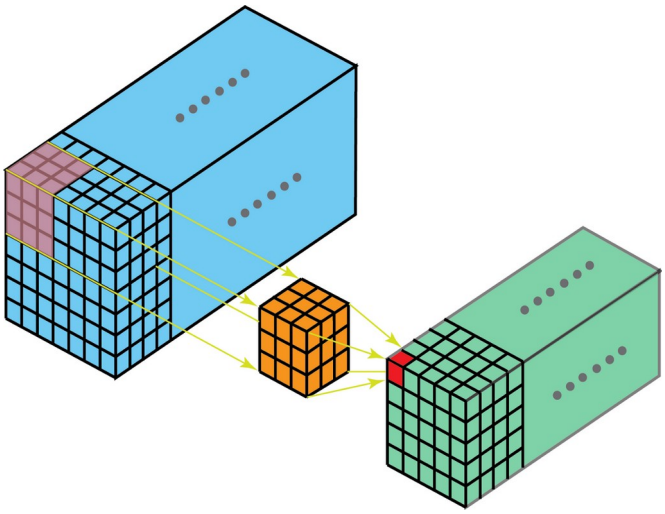
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 32)        896

 max_pooling2d (MaxPooling2D  (None, 15, 15, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 13, 13, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 6, 6, 64)         0
 2D)

 conv2d_2 (Conv2D)           (None, 4, 4, 64)          36928

 flatten (Flatten)           (None, 1024)              0

 dense (Dense)               (None, 64)                65600

 dense_1 (Dense)             (None, 10)                650

=================================================================
Total params: 122,570
Trainable params: 122,570
Non-trainable params: 0
```

Slika 5.Prikaz slojeva i parametara modela

Prema Slici 2. se mogu vidjeti koraci kreiranja slojeva od kojih su korišteni konvolucijski -> *Conv2D* sloj, te sloj udruživanja -> *MaxPooling2D*, uz FCN slojeve. Naime konvolucijski sloj kao ulazne parametre ima brojke 32 i 64 koji ovise o veličini ulazne slike koja je prema CIFAR-10

skupu podataka dimenzija 32x32. Potom su definirani filteri (3,3) koje predstavljaju matrice čija je funkcija obilaska po matričnom prikazu slike, pri tom radeći računske operacije konvolucije nad vrijednostima piksela (Slika 6.).

Kao aktivacijska funkcija korištena je *ReLU* funkcija, kako bi se pronašla nelinearnost unutar



Slika 6.Slikoviti prikaz rada filtera za konvoluciju

podataka. Radi na način da vraća natrag ulaznu vrijednost ako je ta vrijednost veća od 0, ili od nekog definiranog praga, dok 0 vraća ako je ulazna vrijednost negativna.

MaxPooling2D predstavlja *downsampling* filter. Kao i konvolucijski sloj ima filter, no kod ovog sloja je to nešto jednostavnije. Razlog u tomu leži jer ovaj sloj reducira matrice u jednu vrijednost ovisno o odabranoj veličini. U ovom slučaju korišten je filter (2,2) koji će 2x2 matricu reducirati u u 1x1 vrijednost. Ovaj filter reducira matricu, ali zadržava glavne značajke ulaznih vrijednosti.

Flatten sloj konvertira vrijednosti slika u 1D vektor.

Dense slojevi predstavljaju umjetnu neuronsku mrežu, koja računa vjerojatnost da objekt iz slike pripada određenoj klasi.

Broj parametara mreže dobiva se pomoću formule (1)[2]:

$$\boldsymbol{P_n} = [(\boldsymbol{a} \times \boldsymbol{b} \times \boldsymbol{c}) + \mathbf{1}] \times \boldsymbol{d} \qquad (\mathbf{1})$$

U navedenoj formuli su vidljivi različiti parametri od kojih predstavlja broj parametara određenog sloja mreže, **a × b** predstavljaju filtere (za konvoluciju i pridruživanje), a **c** i **d** su ulazno/izlazne jedinica koje predstavljaju veličine mapa značajki (ulaznih/izlaznih matrica slike) unutar konvolucijskog sloja. Konstante koje se dodaju u produkt značajki i težinskih vrijednosti odnosno

(engl. Biases) su kompenzirane u obliku jedinične vrijednosti koja se dodaje unutar zagrada u formuli (1). Također, dimenzije izlazne matrice nakon filtriranja pri izlasku iz konvolucijskog sloja i sloja pridruživanja, predstavljene su varijablom $\mathbf{O_n}$ u formuli (2):

$$On = \frac{N - F}{S} + 1 \qquad (2)$$

Prema formuli (2) $\boldsymbol{N}$ predstavlja ulazne dimenzije, $\boldsymbol{F}$ se odnosi na dimenzije filtera, dok S predstavlja razinu sažimanja podataka (engl. stride). Slojevi pridruživanja nemaju izričitih parametara kao kod konvolucije, iz razloga što računaju srednju ili maksimalnu vrijednost koja se dobiva kroz filter, tako da nema potrebe za optimizacijom navedenog sloja [3].

2.3. Treniranje modela

Nakon izgradnje modela kroz definiranje slojeva mreže, parametara i cjelokupne arhitekture potrebno je model trenirati na datom skupu podataka CIFAR-10, koristeći *training data -> (trainX)* skup vrijednosti. Prvi korak zahtjeva pokretanje modela (engl. compile), unutar koje se koristi funkcija *model.compile()* (Slika 7.), gdje kao argumente je potrebno predati parametre:

- Funkcija rasipanja (*Loss function)*
- Optimizator (*Optimizer)*
- *Metriku (Metrics)*

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

Slika 7.Pokretanje (compiling) modela

Funkcija rasipanja predstavlja način vrednovanja kreiranog modela, odnosno algoritma koji klasificira objekte. Naime, za odabir funkcije rasipanja postoje mogućnosti gdje je moguće odabrati („*Categorical cross entropy”*, *„Binary Cross Entropy”*, *„Sparse Categorical Cross Entropy”*)[8] ovisno o skupu podataka i broju klasa. U ovom radu je korištena funkcija *„Sparse Categorical Cross Entropy”* zbog toga što svaki objekt pripada isključivo jednoj klasi, te nema preklapanja, dok kategorijski znači da je u domeni više od 2 klase.

Optimizator predstavlja funkciju koja oponaša optimizacijski algoritam zamjene atributa neuronske mreže poput težinskih vrijednosti ili brzine učenja. Postoje razni optimizatori poput *Adama*,

AdaDelta, *SGD* itd. *Adam* optimizator je odabran iz razloga što se može koristiti umjesto stohastičke silazne funkcije gradijenta koja iterativnom metodom ažurira težinske vrijednosti neuronske mreže[8]. No, *Adam* to radi uz drugačije performanse poput manjih memorijskih zahtjeva, manje zahtjevnih računskih operacija i sl.

Metrika predstavlja jedan od mjeritelja performansi modela poput preciznosti, srednje kvadratne pogreške i drugih.

Postavljanjem finih podešavanja koristeći funkciju *compile()*, potrebno je izvršiti treniranje modela. Trening modela se radi kroz funkciju *model.fit()*, prikazanu na Slici 8. Naime, treniranje modela kroz funkciju prima određene argumente od kojih prvo moramo specifirati skupove podataka za trening i testiranje, zatim epohe odnosno broj iteracija kroz koje model prolazi, te veličinu kontejnera unutar kojih će model prolaziti kroz slike i učiti. Unutar argumenata se može predati validacijski skup podataka koji se može odvojiti kod koraka raspodjele podataka. No, u ovom slučaju će se koristiti testni skup kao skup za validaciju, čija funkcija bi se ogledala u finom podešavanju modela tijekom treninga.

```
history = model.fit(train_images, train_labels, epochs=10, validation_data=(test_images, test_labels))
```

```
Epoch 1/10
1563/1563 [==============================] - 18s 12ms/step - loss: 0.5444 - accuracy: 0.8087 - val_loss: 0.8901 - val_accuracy: 0.7099
Epoch 2/10
1563/1563 [==============================] - 18s 11ms/step - loss: 0.5108 - accuracy: 0.8216 - val_loss: 0.9165 - val_accuracy: 0.7107
Epoch 3/10
1563/1563 [==============================] - 18s 11ms/step - loss: 0.4823 - accuracy: 0.8300 - val_loss: 0.9166 - val_accuracy: 0.7157
Epoch 4/10
1563/1563 [==============================] - 18s 11ms/step - loss: 0.4465 - accuracy: 0.8423 - val_loss: 0.9316 - val_accuracy: 0.7188
Epoch 5/10
1563/1563 [==============================] - 18s 11ms/step - loss: 0.4139 - accuracy: 0.8541 - val_loss: 0.9614 - val_accuracy: 0.7108
Epoch 6/10
1563/1563 [==============================] - 18s 12ms/step - loss: 0.3926 - accuracy: 0.8609 - val_loss: 0.9754 - val_accuracy: 0.7174
Epoch 7/10
1563/1563 [==============================] - 19s 12ms/step - loss: 0.3657 - accuracy: 0.8691 - val_loss: 1.0807 - val_accuracy: 0.7089
Epoch 8/10
1563/1563 [==============================] - 18s 12ms/step - loss: 0.3458 - accuracy: 0.8772 - val_loss: 1.1038 - val_accuracy: 0.7054
Epoch 9/10
1563/1563 [==============================] - 18s 12ms/step - loss: 0.3167 - accuracy: 0.8874 - val_loss: 1.1142 - val_accuracy: 0.7125
Epoch 10/10
1563/1563 [==============================] - 20s 13ms/step - loss: 0.3023 - accuracy: 0.8923 - val_loss: 1.1696 - val_accuracy: 0.7068
```

Slika 8.Treniranje modela konvolucijske neuronske mreže

3. Rezultati testiranja

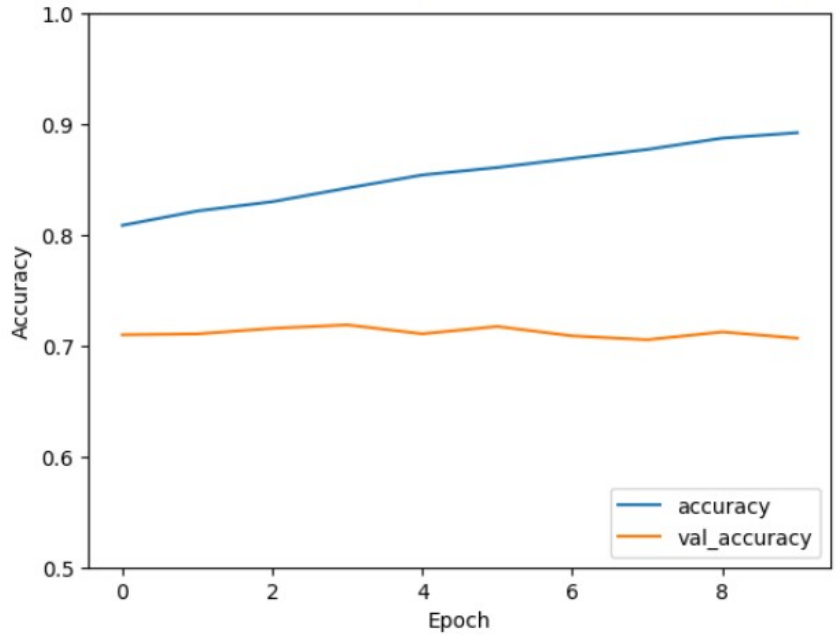
Dobiveni istrenirani model sačinjen od 8 slojeva unutar kojih su postavljeni alternirajućim redom slojevi konvolucije (*CONV*) i pridruživanja (*POOL),* je potrebno testirati i vrednovati. Naime, veličina filtera kod slojeva *CONV* i *POOL* (3x3) i (2x2), te stupnja sažimanja podataka (engl. stride) veličine 2, pokazala bi se kao dobra formulacija za dobivanje solidnih rezultata pri detekciji. Uz to korištenje *ReLU* aktivacijske funkcije predstavlja dobar odabir iz razloga što nije osjetljiva na promjenu gradijenta, koji se odnosi na problem prilikom povratne vrijednosti aktivacijskih funkcija.

Međutim, iako *Softmax* aktivacijska funkcija ima problema s nestajućim (promjenjivim) gradijentom koji može dovesti do nestabilnosti mreže jer zna prouzročiti nefunkcioniranje neurona (vrijednost 0.0), takva funkcija pruža dobru vjerojatnosnu raspodjelu odluka (izlaznih vrijednosti) za više izlaznih vrijednosti te se postavlja kao izlazna aktivacijska funkcija kod zadnjeg sloja mreže. Naime, *Softmax* omogućuje donošenje odluke na osnovu najveće vjerojatnoće da se neki objekt nalazi primjerice na slici i pripada određenoj klasi, tako što izlazne vrijednosti zbraja i dovodi ih do vrijednosti jedan.

Kako bi vrednovali model potrebno je bilo koristiti *matplotlib* biblioteke, čime bi se konačni rezultat preciznosti i funkcije gubitaka prikazali. Vrednovanje modela se provodi kroz funkciju *model.evaluate()*, koja kao rezultat prikazuje vrijednost i kretanja performanse preciznosti (Slika 9.)

```
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')

test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)
```

```
313/313 - 1s - loss: 1.1696 - accuracy: 0.7068 - 1s/epoch - 4ms/step
```



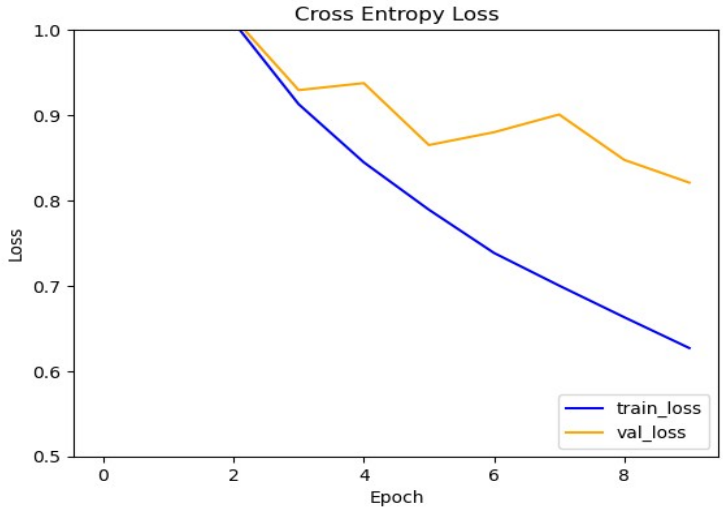
Slika 9.Kretanja preciznosti tijekom treninga i validacije

Naime, iz Slike 9. se može vidjeti da se tijekom treniranja odnosno kroz epohe povećava preciznost modela, te da ona dolazi do solidnih vrijednosti od 0.9. Dakako se također može vidjeti povećavanje razlike između preciznosti tijekom treniranja i validacije kako se broj iteracija povećava. Ovo može dovesti model do *overfitt-a* ako povećavamo broj iteracija, odnosno preciznost validacije će opadati

dok treninga će rasti. Stoga se može reći da za trenutne postavke model pruža dobre rezultate pri odlučivanju. S druge strane postoji mogućnost i da se preciznost validacije zadrži na nekih 70% što je uredan rezultat.

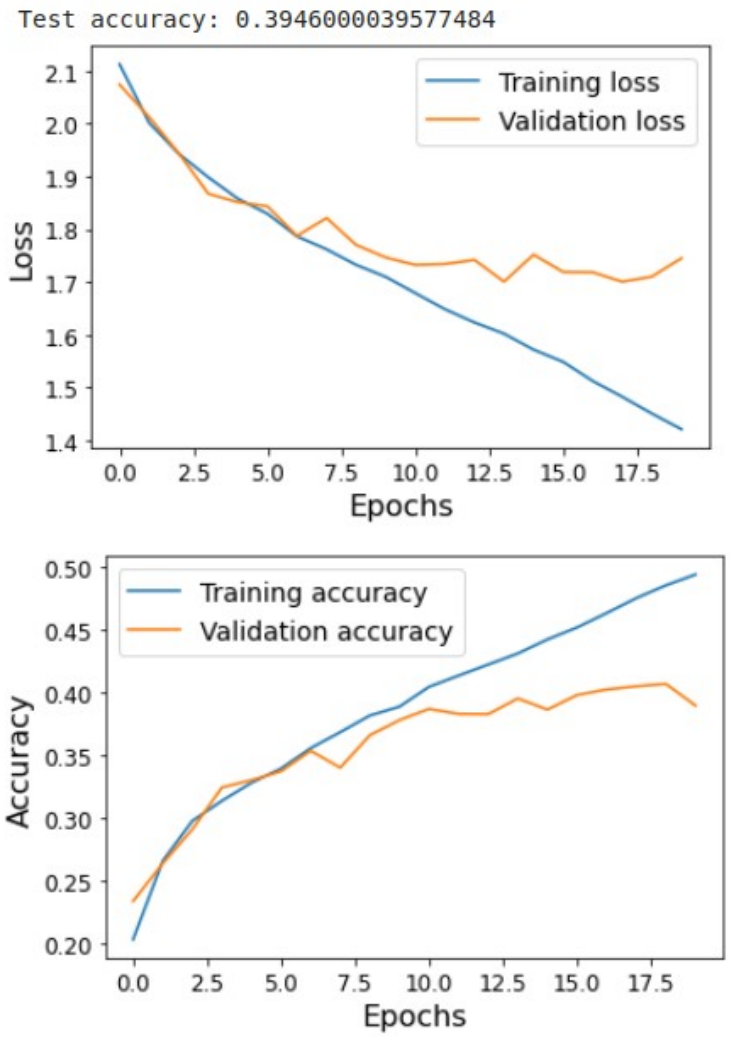
Također, pored preciznosti su prikazane i kretanja vrijednosti gubitaka (Slika 10).

```
In [13]: plt.plot(1)
         plt.title('Cross Entropy Loss')
         plt.plot(history.history['loss'], color='blue', label='train_loss')
         plt.plot(history.history['val_loss'], color='orange', label='val_loss')
         plt.xlabel('Epoch')
         plt.ylabel('Loss')
         plt.ylim([0.5, 1])
         plt.legend(loc='lower right')
         # plot accuracy

         #plt.subplot(212)
         ##plt.title('Classification Accuracy')
         #plt.plot(history.history['accuracy'], color='blue', label='train')
         #plt.plot(history.history['val_accuracy'], color='orange', label='test')
Out[13]: <matplotlib.legend.Legend at 0x7f567c0dbd90>
```



Slika 10.Vrijednosti gubitaka tijekom treniranja i validacije kroz epohe

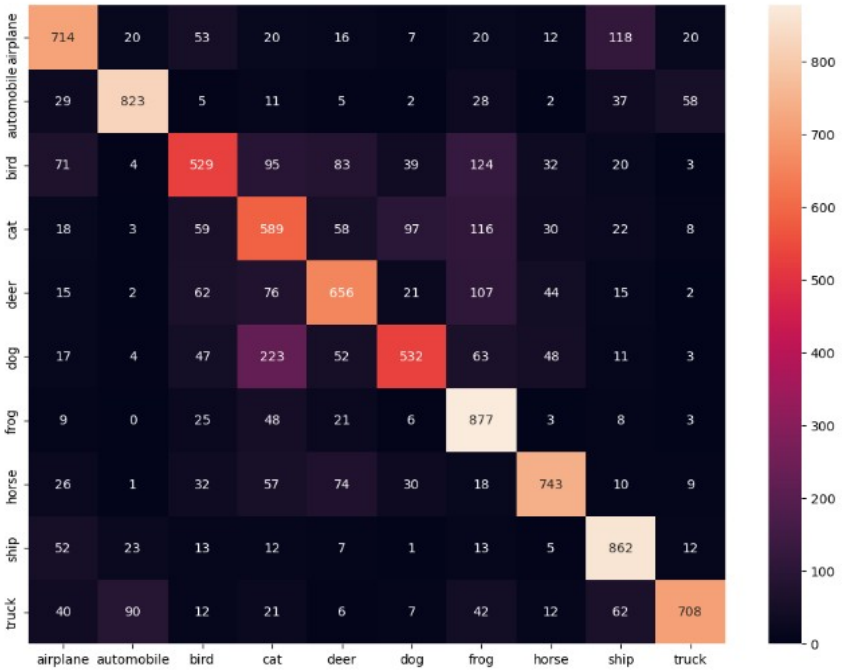
Prema Slici 10. se može vidjeti kako vrijednosti gubitaka treninga i validacije znatno opadaju, što je analogno tomu da se preciznost povećava. Ova informacija govori o tome koliko su određena predviđanja dobro provedena. Vidljivo je da povećanjem iteracija gubitak se značajnije smanjuje. Drugim riječima, sve je manje penala koje model prima zbog krivih predviđanja. Između ostalog ovaj model se može usporediti s model iz primjera s potpuno povezanom neuronskom mrežom gdje su rezultati preciznosti dostigle manje vrijednosti Slika 11. Iz sljedeće slike se također može uvidjeti da čak i jedan sloj konvolucijske mreže igra značajnu ulogu pri radu s klasifikacijom objekata iz slika. Dakako se može vidjeti potencijal konvolucijskih neuronskih mreža, osobito kod grafova s prikazom preciznosti (točnih detekcija), gdje vrijednosti su značajno veće nego kod obične neuronske mreže.



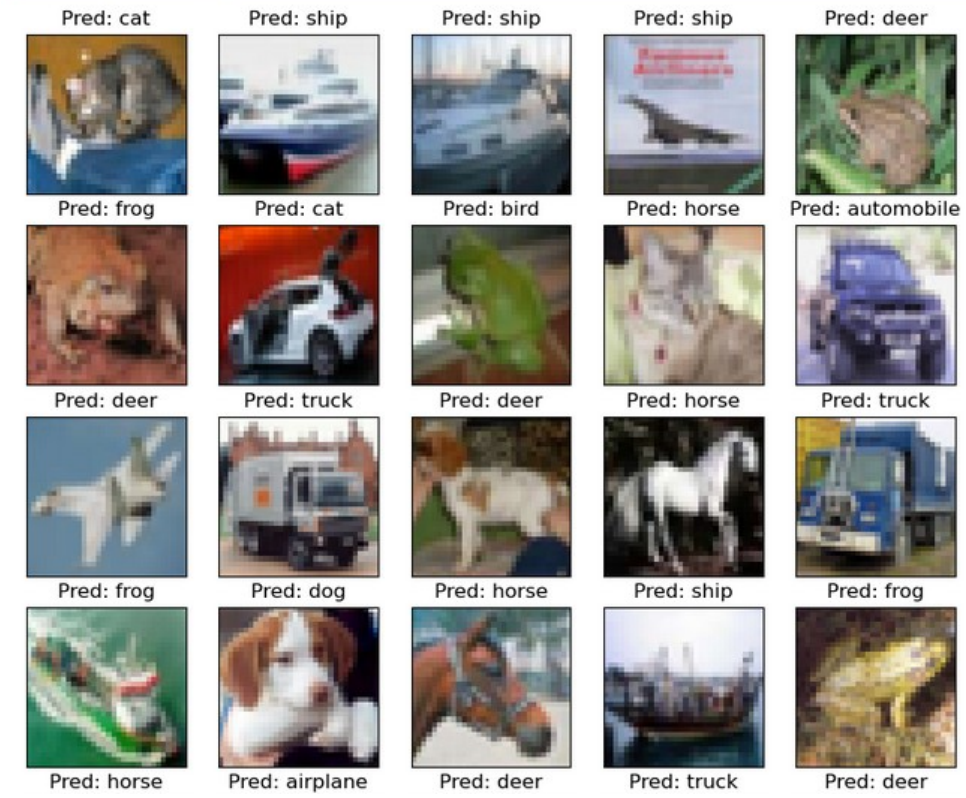
Slika 11.Prikaz vrednovanja modela za umjetnu neuronsku mrežu (FCN)[4]

Pored navedenih vrednovanja i grafova za ocjenjivanje modela također je na Slici 12. prikazana matrica konfuzije. Prema matrici konfuzije se mogu vidjeti predviđanja modela, gdje pak najveće vrijednosti imaju pozicije matrica gdje se imena oznaka s osi apcise i ordinate podudaraju. Osim toga, u ostalim pozicijama matrice postoje određena odstupanja zbog krivih predviđanja, no može se ustanoviti da model uspijeva detektirati objekte koji se nalazi u slici s većom preciznošću. To se također može vidjeti na pojednostavljenom matričnom prikazu slika (Slika 13.). Uz navedene slike su prikazane i oznake s stvarnim i predviđenim vrijednostima.

```
 Text(7.5, 0, 'horse'),
 Text(8.5, 0, 'ship'),
 Text(9.5, 0, 'truck')],
[Text(0, 0.5, 'airplane'),
 Text(0, 1.5, 'automobile'),
 Text(0, 2.5, 'bird'),
 Text(0, 3.5, 'cat'),
 Text(0, 4.5, 'deer'),
 Text(0, 5.5, 'dog'),
 Text(0, 6.5, 'frog'),
 Text(0, 7.5, 'horse'),
 Text(0, 8.5, 'ship'),
 Text(0, 9.5, 'truck')]]
```



Slika 12.Matrica konfuzije za model CNN (CIFAR-10)



Slika 13.Rezultati predviđanja istreniranog modela nad testnim skupom podataka

4. Implementacija modela kroz mobilnu aplikaciju

Finalni model konvolucijske neuronske mreže dodatno se može proširiti s implementacijom na drugi vanjski uređaj poput pametnog mobitela, ugradbenog sustava i sl. U ovom projektu će istrenirani model biti implementiran kroz mobilnu aplikaciju koristeći *Android Studio.* Prije toga je potrebno konvertirati model koristeći *Tensorflow Lite* mobilnu biblioteku namijenjenu za rad na prijenosnim manjim uređajima. Konverzija i spremanje modela za učitavanja unutar *Android Studio* je prikazano na Slici 14.

```
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

with open("model.tflite", 'wb') as f:
  f.write(tflite_model)
```

Slika 14.Konverzija Keras modela u Tensorflow Lite model

Konvertirani *Tensorflow Lite* model je potrebno spremiti unutar repozitorija gdje će aplikacija biti kreirana. Uostalom, ideja izvedbe i testiranja modela kroz mobilnu aplikaciju ogleda se u tome da se kroz XML kreiraju dugmići, preko kojih će se moći odabrati opcija kojom će se klasifikacija odvijati. Postoje opcije aktivnog detektiranja predmeta, odnosno slikanjem, te opcija učitavanja postojeće slike iz galerije. Dodatna stavka su tekstualni okviri unutar kojih bi se slao rezultat detekcije. Kao bonus opcija su dodane vrijednosti vjerojatnosne distribucije koje dodatno opisuju kojoj klasi pripada objekt na slici ukoliko ga ima.

Glede *backend* dijela, potrebno je napraviti arhitekturu koja bi odgovarala onoj arhitekturi kreiranog modela u *Keras-u.* Stoga, prvi korak je bio sastavljen u kreiranju ulaza za referentne slike gdje se definirao spremnik unutar kojeg se spremaju pikseli slike. Potom, ključni faktor za razvoj sljedećih koraka je postavka dimenzija slike, što će omogućiti kasnije popunjavanje dodatnog spremnika podataka. Dodatni *byteBuffer* predstavlja spremnik unutar kojeg se dodaju pikseli ulazne slike i spremaju prema bojama (*RGB*). To je jedna osnova za daljnjo procesiranje slike, jer će se kao ulazna značajka učitavati pikseli iz kreiranog spremnika, te procesirati od strane modela. Glavna funkcija koja obavlja procesiranje i obradu slike s poveznicom na model je *model.process(inputFeature0).* Takva funkcija ima povratnu vrijednost unutar koje se nalazi krajnja odluka koju je potrebnu izbrusiti korištenjem pomoćnih *Tensorflow Lite* funkcija gdje će se kao

povratna vrijednost dobiti vjerojatnost da se nalazi određeni objekt na slici. U konačnici je potrebno definirati klase, te algoritmom provjere najveće vrijednosti pronaći indeks na kojoj se ta vrijednost i ispisati ju kao konačnu odluku. Izvedbeni kod je prikazan na Slici 15.

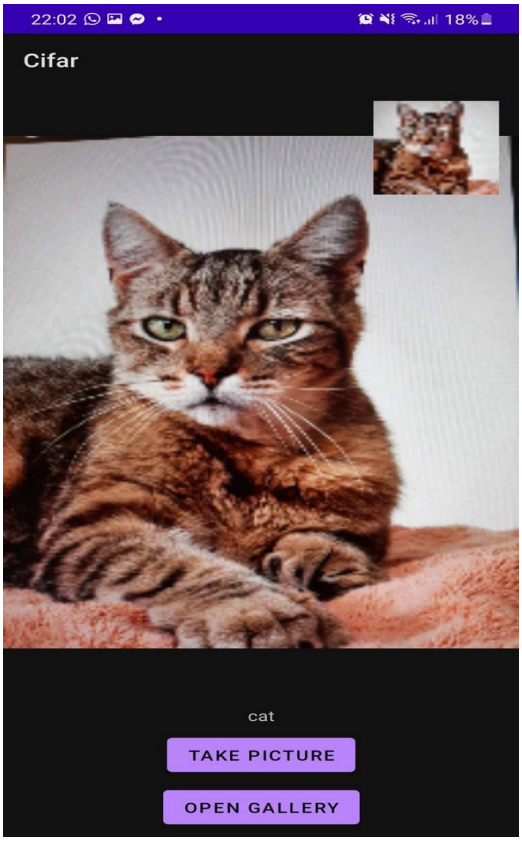
```
try {
    val model = Model.newInstance(applicationContext)

    // Creates inputs for reference.
    val inputFeature0 =
        TensorBuffer.createFixedSize(intArrayOf(1, 32, 32, 3), DataType.FLOAT32)
    val byteBuffer: ByteBuffer = ByteBuffer.allocateDirect(4 * imageSize * imageSize * 3)
    byteBuffer.order(ByteOrder.nativeOrder())
    val intValues = IntArray(imageSize * imageSize)
    image.getPixels(intValues, 0, image.width, 0, 0, image.width, image.height)
    var pixel = 0
    //iterate over each pixel and extract R, G, and B values. Add those values individually to the byte buffer.
    for (i in 0 until imageSize) {
        for (j in 0 until imageSize) {
            val `val` = intValues[pixel++] // RGB
            byteBuffer.putFloat((`val` shr 16 and 0xFF) * (1f / 1))
            byteBuffer.putFloat((`val` shr 8 and 0xFF) * (1f / 1))
            byteBuffer.putFloat((`val` and 0xFF) * (1f / 1))
        }
    }
    inputFeature0.loadBuffer(byteBuffer)

    // Runs model inference and gets result.
    val outputs = model.process(inputFeature0)
    val outputFeature0 = outputs.outputFeature0AsTensorBuffer
    val confidences = outputFeature0.floatArray
    // find the index of the class with the biggest confidence.
    var maxPos = 0
    var maxConfidence = 0f

    val classes = arrayOf("airplane", "automobile", "bird", "cat", "deer",
        "dog", "frog", "horse", "ship", "truck")

    var output = ""
    for (i in confidences.indices) {
        if (confidences[i] > maxConfidence) {
            maxConfidence = confidences[i]
            maxPos = i
        }

        output += classes[i] + ": " + confidences[i] + "\n"

    }
```

Slika 15.Izrada strukture u Kotlin-u, za klasifikaciju objekta iz slike koristeći TensorFlow Lite

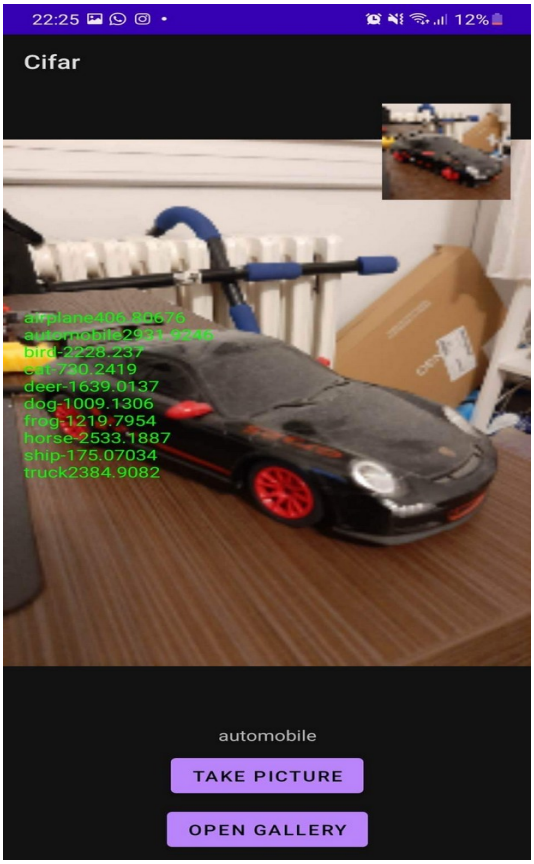
4.1. Rezultati testiranja

Nakon provedbe validacije i verifikacije kreirane aplikacije, idući korak je bio u testiranju donesenih odluka i vrednovanju novog konvertiranog modela i njegov rad kroz mobilnu aplikaciju. U prvim slikama su izostavljeni vrijednosni opisi vjerojatnosni.

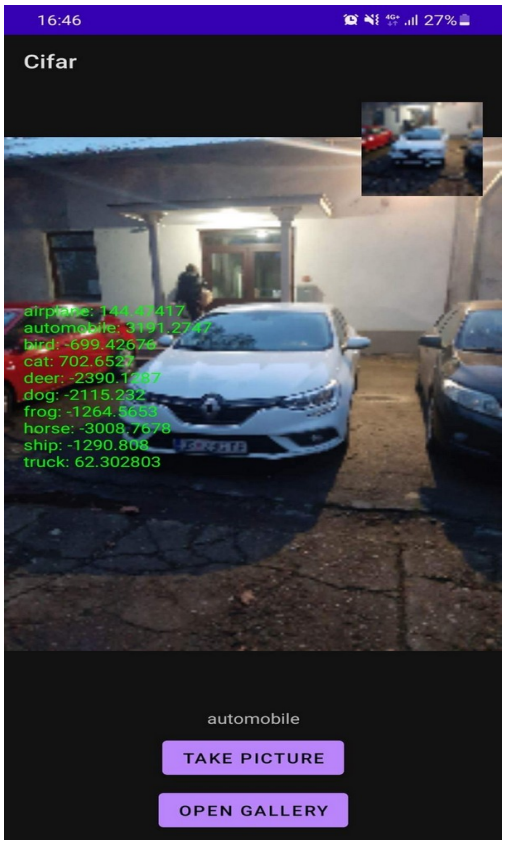
Rezultati testiranja su prikazani na sljedećim slikama:



Slika 16.Testiranje modela kroz aplikaciju – detekcija mačke



Slika 17.Testiranje modela - detekcija automobila



Slika 18.Testiranje modela - detekcija automobila

5. Zaključak

Fokus ovog projekta je bio u izradi i testiranju konvolucijskih neuronskih mreža na CIFAR-10 skupu podataka. Izbor podataka za treniranje i testiranje je proveden kroz CIFAR-10, iz razloga što se činio adekvatnim i uredno segmentiranim skupom podataka s 10 klasa. Svaki objekt s slike je pripadao jednoj klasi bez preklapanja, te je za rad s ovim podacima bila potrebna normalizacija i podjela na trening i test podskup. Potom je definiran model konvolucijske neuronske mreže u kojoj se odabrani konvolucijski slojevi i slojevi pridruživanja kako bi konačni rezultat bio veće preciznosti. Kao optimalna arhitektura odabrana je mreža s 8 slojeva i s adekvatnim filterima koji su u koraku vrednovanja i testiranja dali uredne rezultate. Između ostalog kod vrednovanja je uočeno da je funkcija preciznosti postigla znatno bolje rezultate od funkcija gubitaka što je svjedočilo o kvaliteti modela. U konačnici je model konvertiran za potrebe mobilne aplikacije, te implementiran na istu. Model je testiran kroz kameru mobitela, te su dati potrebni vjerojatnosni opisi konačnih odluka.

LITERATURA

1. V. Flovik, Handling overfitting in deep learning models, 2018. Accessed on: Oct. 9, 2021[Online]. Available: https://towardsdatascience.com/handling-overfitting-in-deep-learning-models-c760ee047c6e
2. S. Ayala, Convolutional Neural Network Implementation for Image Classification using CIFAR-10 Dataset, Techical Report Studeni 2021, Norfolk State University, 700 Park Avenue, Norfolk, USA, 23504
3. A. D. Akwaboah Convolutional Neural Network for CIFAR-10 Dataset Image Classification, Studeni 2019, Johns Hopkins University
4. Github Repozitorij - https://github.com/Adakwaboah/CIFAR10_Neural_Network_image_classification/blob/master/cifar10_NN_Akwasi.ipynb

5. F. O. Giuste, CIFAR-10 IMAGE CLASSIFICATION USING FEATURE ENSEMBLES, iomedical Engineering, Georgia Institute of Technology, Atlanta, GA, USA, 2020

6. J. Brownlee, Deep Learning for Computer Vision, https://machinelearningmastery.com/how-to-develop-a-cnn-from-scratch-for-cifar-10-photo-classification/

7. A. Brahame, Deep Learning with CIFAR-10 https://towardsdatascience.com/deep-learning-with-cifar-10-image-classification-64ab92110d79, 2020

8. D. Madhugiri, Using CNN for Image Classification on CIFAR-10 Dataset https://devashree-madhugiri.medium.com/using-cnn-for-image-classification-on-cifar-10-dataset-7803d9f3b983, 5. Kolovoz, 2021