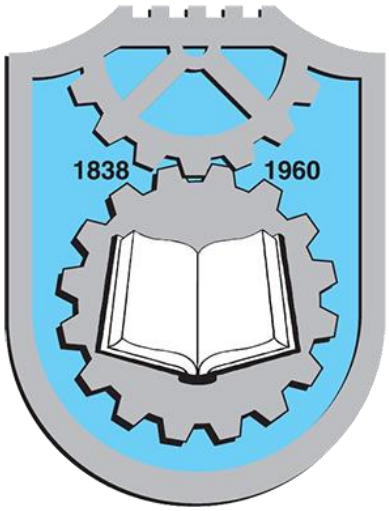
Univerzitet u Kragujevcu
Fakultet inženjerskih nauka



Predmet:
Osnovi dubokog učenja

Tema: Deep Dream

Student:
Anđela Manojlović 628/2018

Predmetni nastavnik:
Vladimir Milovanović

Kragujevac, jun 2022. godine

Sadržaj

1.Opis zadatka

Tema ovog rada je generisanje nove slike na osnovu postojeće. Izlaz iz mreže (nova slika) predstavlja „deformisan” ulaz. Do toga dolazi tako što neuronska mreža pokušava da prepozna objekte na slici, ali u dubljim slojevima nejasne delove pokušava da predstavi onim koji su joj već poznati. Izlazne slike su tako pre-procesirane. Ovaj koncept je zasnovan na kompjuterskoj viziji DeepDream kompanije Google. U zavisnosti od toga da li se slika dodatno modifikuje, koji slojevi i karakteristike se maksimizuju, I da li se dodaje tzv. „target” slika, implementacije ove ideje mogu varirati. Primer rada DeepDream mreže kompanije Google dat je na slici ispod.

(Slika 1) Ulazne i izlazne slike mreže DeepDream kompanije Google

2.Korišćene tehnologije

Za realizaciju ove mreže korišćen je Python programski jezik, pisan u IDLE razvojnom okruženju, kao i u Jupyter interaktivnoj radnoj svesci radi bržih izračunavanja.

Korišćene biblioteke su sledeće:

- **TensorFlow** je biblioteka otvorenog koda za mašinsko učenje I veštačku inteligenciju. Može se koristiti u nizu zadataka, alii ma poseban fokus na obuci i zaključivanju dubokih neuronskih mreža. TensorFlow je razvio tim Google Brain za internu upotrebu kompanije Google u istraživanju i u proizvodnji.

- **PIL**(Python Imaging Library) je besplatna dodatna biblioteka otvorenog koda za programski jezik Python koja dodaje podršku za otvaranje, manipulisanje I čuvanje mnogih različitih formata datoteka slika.

- **Numpy** je proširenje na Python programskom jeziku,dodajući podršku za velike, višedimenzionalne nizove i matrice, zajedno sa velikim bibliotekama matematičkih funkcija na visokom nivou koje mogu manipulisati nad ovim nizovima.

- **Matplotlib** je biblioteka za programski jezik Python i služi za grafički prikaz I vizuelizaciju podataka. Otvorenog je koda, uglavnom pisana u Python-u. Najčešće se koristi njen podmodul pyplot koji se obično uvozi pod aliasom plt.

- **Keras** je biblioteka otvorenog koda za komponente neuronske mreže pisana u Pythonu. Obezbeđuje aktivacione funkcije,optimizacije, podršku za rekurentne I konvolucijske neuronske mreže, I druge funkcionalnosti korisne za oblast mašinskog učenja I veštačke inteligencije.

Klasifikacioni model neuronske mreže je importovan iz Keras biblioteke, učitan sa težinskim koeficijentima iz već trenirane mreže ImageNet.

3.Opis koda

Nakon što su sve potrebne biblioteke importovane, potrebno je pronaći sliku koju će mreža transformisati, u .jpg formatu i njenu URL adresu sačuvati u promenljivoj url. Nakon toga sledi preuzimanje slike sa date adrese i njena obrada.

```
In [14]: import tensorflow as tf
         import numpy as np
         import matplotlib as mpl
         import IPython.display as display
         import PIL.Image

In [15]: url= 'https://time.com/wp-content/uploads/2017/10/229-westerlund-21.jpg'
```

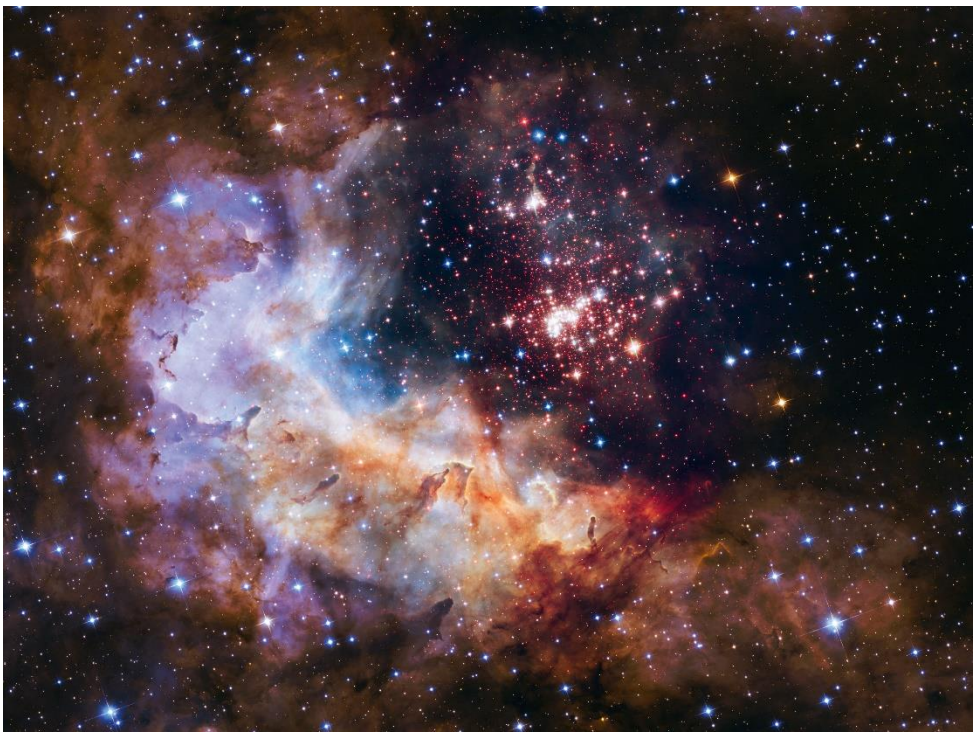
(Slika 2) Uvoženje biblioteka

Prva funkcija u kodu je funkcija download() koja preuzima sliku sa URL adrese navedene u promenljivoj url, i vraća istu kroz numpy niz. Ovaj niz sadrži vrednosti od 0 do 255 za odgovarajući kanal R, G i B svakog piksela slike. Nakon što je slika preuzeta, koriguju se dimenzije kako bi rad sa slikom bio olakšan, i prikazuje se originalna slika pomoću funkcije show().

```
In [16]: # Download an image and read it into a NumPy array.
         def download(url, max_dim=None):
           name = url.split('/')[-1]
           image_path = tf.keras.utils.get_file(name, origin=url)
           img = PIL.Image.open(image_path)
           if max_dim:
             img.thumbnail((max_dim, max_dim))
           return np.array(img)

         # Normalize an image
         def deprocess(img):
           img = 255*(img + 1.0)/2.0
           return tf.cast(img, tf.uint8)

         # Display an image
         def show(img):
           display.display(PIL.Image.fromarray(np.array(img)))


         # Downsizing the image makes it easier to work with.
         original_img = download(url, max_dim=500)
         show(original_img)
         display.display(display.HTML('Image cc-by: <a "href=https://commons.wikimedia.org/wiki/File:Felis_catus-cat_on_snow.jpg">Von.grz
```

(Slika 3) *Preuzimanje i priprema slike*

Originalna slika čija je URL adresa navedena u kodu prikazana je na slici ispod radi kasnijeg upoređivanja sa slikom koja predstavlja izlaz iz ovakve mreže:



(Slika 4) *Originalna slika korišćena kao ulaz u mrežu*

Sledeći važan korak je učitavanje modela. Korišćenjem dolenavedene funkcije iz Keras biblioteke, preuzima se već istreniran model za klasifikaciju slika. Konkretno, koristi se InceptionV3 model konvolucijske neuronske mreže koji je inače veoma sličan modelu korišćenom u originalnoj realizaciji ove ideje. Moguće je ovaj model zameniti bilo kojim drugim, već istreniranim modelom.

```
In [17]: base_model = tf.keras.applications.InceptionV3(include_top=False, weights='imagenet')
```

(Slika 5) Preuzimanje InceptionV3 modela

Ideja realizacije ovakve mreže je da se izabere neki sloj (ili više njih) i da se maksimizuje njihov gubitak na takav način da slika intenzivno pobuđuje slojeve. Funkciju gubitaka koristimo da odredimo grešku,tj. razliku između željenog izlaza i dobijenog izlaza. Ta greška naziva se gubitak. Složenost karakteristika mreže zavisi od slojeva odabranih da se njihovi gubici

maksimizuju, tj. niži slojevi proizvode linije ili jednostavne šare, dok dublji slojevi daju neke jasnije karakteristike slika, ili čak cele objekte.

Arhitektura InceptionV3 mreže je vrlo velika. Za DeepDream, slojevi koji su od interesa su oni gde su konvolucije povezane. Ova mreža sadrži 11 ovakvih povezanih slojeva, i označeni su sa “mixed0”, “mixed1” do “mixed10”. Korišćenje različitih slojeva rezultiraće različitim izlaznim slikama. Dublji slojevi odgovaraju karakteristikama višeg nivoa (npr.lica, oči), dok raniji slojevi odgovaraju jednostavnijim obrascima. U daljem kodu, maksimizovane su aktivacione funkcije slojeva “mixed3” i “mixed5”, a moguće je zameniti ih bilo kojim drugim, imajući u vidu da dublji slojevi iniciraju duže treniranje, zbog složenijih izračunavanja.

In [18]:
```
# Maximize the activations of these layers
names = ['mixed3', 'mixed5']
layers = [base_model.get_layer(name).output for name in names]

# Create the feature extraction model
dream_model = tf.keras.Model(inputs=base_model.input, outputs=layers)
```

(Slika 6) Maksimizacija aktivacionih funkcija izabranih slojeva

Gubitak je suma aktivcionih funkcija izabranih slojeva. Gubitak se normalizuje u svakom sloju kako uticaj dubljih slojeva ne bi „zagušio” uticaj ranijih slojeva. Inače, gubitak je veličina koju je potrebno minimizovati pomoću gradijentnog spusta. Za realizaciju DeepDream mreže, ovu veličinu maksimizujemo pomoću gradijentnog uspona.

In [19]:
```
def calc_loss(img, model):
  # Pass forward the image through the model to retrieve the activations.
  # Converts the image into a batch of size 1.
  img_batch = tf.expand_dims(img, axis=0)
  layer_activations = model(img_batch)
  if len(layer_activations) == 1:
    layer_activations = [layer_activations]

  losses = []
  for act in layer_activations:
    loss = tf.math.reduce_mean(act)
    losses.append(loss)

  return  tf.reduce_sum(losses)
```

(Slika 7) Računanje gubitka

Nakon završenog računanja gubitaka, za minimalnu implementaciju DeepDream mreže potrebno je samo još izračunati gradijente I dodati ih

originalnoj slici. Dodavanje gradijenata originalnoj slici pojačava intenzitet obrazaca koje je mreža uočila. U svakom koraku, slika sve više pobuđuje aktivacione funkcije pojedinih slojeva. Za realizaciju ovoga odgovorna je funkcija DeepDream, koja je obmotana u tf.function() radi boljih performansi i izbegavanja pojedinih grešaka. Koristi input_signature da osigura da funkcija nije ometena drugim dimenzijama slika ili koracima/ veličinama koraka.

```
In [20]: class DeepDream(tf.Module):
           def __init__(self, model):
             self.model = model

           @tf.function(
               input_signature=(
                 tf.TensorSpec(shape=[None,None,3], dtype=tf.float32),
                 tf.TensorSpec(shape=[], dtype=tf.int32),
                 tf.TensorSpec(shape=[], dtype=tf.float32),)
           )
           def __call__(self, img, steps, step_size):
               print("Tracing")
               loss = tf.constant(0.0)
               for n in tf.range(steps):
                 with tf.GradientTape() as tape:
                   # This needs gradients relative to `img`
                   # `GradientTape` only watches `tf.Variable`s by default
                   tape.watch(img)
                   loss = calc_loss(img, self.model)

                 # Calculate the gradient of the loss with respect to the pixels of the input image.
                 gradients = tape.gradient(loss, img)

                 # Normalize the gradients.
                 gradients /= tf.math.reduce_std(gradients) + 1e-8

                 # In gradient ascent, the "loss" is maximized so that the input image increasingly "excites" th
                 # You can update the image by directly adding the gradients (because they're the same shape!)
                 img = img + gradients*step_size
                 img = tf.clip_by_value(img, -1, 1)

               return loss, img
```

(Slika 8) Računanje gubitka

Pozivanjem sledeće funkcije kreira se model.

```
In [21]: deepdream = DeepDream(dream_model)
```

(Slika 9) Kreiranje modela

Nakon što je model kreiran, DeepDream mreža se može demonstrirati i unaprediti na različite načine. U nastavku biće prikazan jednostavan rad pomoću funkcije run_deep_dream_simple(), unapređena varijanta koja rešava probleme nastale primenom prve funkcije, kao i rad mreže uz dodatno modifikovanje ulazne slike pre nego što mreža uoči obrasce na slici.

3.1 Jednostavna realizacija

Prvi od tri načina realizacije DeepDream mreže je pomoću sledeće funkcije run_deep_dream_simple(). Osim slike koja se kao parametar prosleđuje u obliku numpy niza, parametri ove funkcije su i broj koraka,kao i veličina koraka.
Najpre se slika,tačnije numpy niz kojim je predstavljena, preprocesira i konvertuje u tensor. U tenzor se konvertuje i veličina koraka (promenljiva step_size). Tenzor predstavlja multidimenzionalni niz uniformnog tipa. Tenzori se ne mogu menjati nakon kreiranja, već se samo može kreirati novi. Navedene promenljive prevedene su u tenzore radi njihove dalje obrade pomoću TensorFlow funkcija.

In [22]:
```
def run_deep_dream_simple(img, steps=100, step_size=0.01):
  # Convert from uint8 to the range expected by the model.
  img = tf.keras.applications.inception_v3.preprocess_input(img)
  img = tf.convert_to_tensor(img)
  step_size = tf.convert_to_tensor(step_size)
  steps_remaining = steps
  step = 0
  while steps_remaining:
    if steps_remaining>100:
      run_steps = tf.constant(100)
    else:
      run_steps = tf.constant(steps_remaining)
    steps_remaining -= run_steps
    step += run_steps

    loss, img = deepdream(img, run_steps, tf.constant(step_size))

    display.clear_output(wait=True)
    show(deprocess(img))
    print ("Step {}, loss {}".format(step, loss))


  result = deprocess(img)
  display.clear_output(wait=True)
  show(result)

  return result
```

(Slika 10) Prva funkcija za pokretanje DeepDream mreže

U svakom koraku, u while petlji, računamo gubitak i modifikujemo sliku, ali se ona zbog funkcije clear_output(wait=True) prikazuje tek u konačnom obliku. Ova funkcija stoji na steku sve dok podaci, u ovom slučaju tensor niz, ne budu spremni, tj.izmenjeni . Kada se prođe kroz sve korake i podaci su spremni, funkcija se automatski odmah poziva pre nego što su novi podaci dodati. Time se obezbeđuje da se stari,obrisani („pregaženi”) podaci ne pojavljuju umesto novih. Rezultat je deprocesirana slika koja se na kraju prikazuje. Za ulaznu sliku, pozivom ove funkcije (poziv funkcije prikazan je na slici 11), dobijamo izlaz prikazan na slici 12. Funkcija se poziva nad originalnom slikom, 100 koraka i veličinom koraka 0.01

```
In [23]: dream_img = run_deep_dream_simple(img=original_img,
                                           steps=100, step_size=0.01)
```

(Slika 11) Poziv funkcije run_deep_dream_simple()

(Slika 12) Rezultat nakon poziva funkcije run_deep_dream_simple()

Pozivom ove funkcije jeste postignut DeepDream efekat ali je slika smanjene rezolucije i obrasci se pojavljuju u prilično sličnoj granulaciji. Ovi nedostaci mogu biti prevaziđeni korišćenjem skaliranja, na način objašnjen u nastavku.

3.2 Realizacija mreže sa prethodnim skaliranjem

Rešenje loše rezolucije i previše ujednačene granulacije dobijenog izlaza iz prethodnog načina realizacije DeepDream mreže može se postići primenom gradijentnog uspona u različitoj meri na različite slojeve. Ovo će omogućiti da se obrasci uočeni na nižim slojevima ugrade u obrasce uočene na dubljim slojevima i time popune izlaznu sliku dodatnim detaljima. Povećanjem vrednosti promenljive SCALE, deformisanja će biti veća, ali i izračunavanja duža. Problem lošije rezolucije može se rešiti tako što će se, nakon primene skaliranja, slika povećati i proces ponoviti u nekoliko oktava. Nakon ove optimizacije štampa se i utrošeno vreme, koje se meri od definisanja skale do prikazivanja izlazne slike.

In [24]:
```
import time
start = time.time()

OCTAVE_SCALE = 1.50

img = tf.constant(np.array(original_img))
base_shape = tf.shape(img)[:-1]
float_base_shape = tf.cast(base_shape, tf.float32)

for n in range(-2, 3):
  new_shape = tf.cast(float_base_shape*(OCTAVE_SCALE**n), tf.int32)

  img = tf.image.resize(img, new_shape).numpy()

  img = run_deep_dream_simple(img=img, steps=50, step_size=0.01)

display.clear_output(wait=True)
img = tf.image.resize(img, base_shape)
img = tf.image.convert_image_dtype(img/255.0, dtype=tf.uint8)
show(img)

end = time.time()
end-start
```

(Slika 13) Primena skaliranja uz gradijentni uspon

Uz primenu skaliranja u vrednosti 1.5, dobijena je sledeća izlazna slika:

(Slika 14) Izlazna slika uz primenu skaliranja

3.3 Realizacija mreže sa prethodnom obradom slike

Još jedna stvar koju treba uzeti u obzir je da, kako se slika povećava, tako će se povećavati i vreme i memorija utrošeni za izračunavanje gradijenta. Primena skaliranja neće raditi na veoma velikim slikama ili na mnogo oktava, a da bi se ovi problem izbegli, slika se može podeliti na pločice (pravougaonike), a gradijent se računati pojedinačno za svaku pločicu. Kako na izlaznoj slici ne bi bile vidljive linije po kojima se pločice spajaju, dobro je pre svakog izračunavanja nasumično pomerati sliku.
Slučajno pomeranje implementirano je pomoću funkcije random_roll()

```
In [25]: #dodatna obrada slike (opciono)
def random_roll(img, maxroll):
  # Randomly shift the image to avoid tiled boundaries.
  shift = tf.random.uniform(shape=[2], minval=-maxroll, maxval=maxroll, dtype=tf.int32)
  img_rolled = tf.roll(img, shift=shift, axis=[0,1])
  return shift, img_rolled

shift, img_rolled = random_roll(np.array(original_img), 512)
show(img_rolled)
```

(Slika 15) Funkcija random_roll() za nasumično pomeranje

Nakon obrade slike dobija se nova slika, koja će biti ulaz u mrežu. Izgled novog ulaza u mrežu prikazan je na slici ispod.

(Slika 16) Ulazna slika nakon obrade slučajnim pomeranjem

Funkcija TiledGradients() ekvivalentna je funkciji deepdream definisanoj ranije, prilagođena slici podeljenoj na proizvoljne pravougaonike (pločice).

In [26]:
```
class TiledGradients(tf.Module):
  def __init__(self, model):
    self.model = model

  @tf.function(
      input_signature=(
        tf.TensorSpec(shape=[None,None,3], dtype=tf.float32),
        tf.TensorSpec(shape=[2], dtype=tf.int32),
        tf.TensorSpec(shape=[], dtype=tf.int32),)
  )
  def __call__(self, img, img_size, tile_size=512):
    shift, img_rolled = random_roll(img, tile_size)

    # Initialize the image gradients to zero.
    gradients = tf.zeros_like(img_rolled)

    # Skip the last tile, unless there's only one tile.
    xs = tf.range(0, img_size[1], tile_size)[:-1]
    if not tf.cast(len(xs), bool):
      xs = tf.constant([0])
    ys = tf.range(0, img_size[0], tile_size)[:-1]
    if not tf.cast(len(ys), bool):
      ys = tf.constant([0])

    for x in xs:
      for y in ys:
        # Calculate the gradients for this tile.
        with tf.GradientTape() as tape:
          # This needs gradients relative to `img_rolled`.
          # `GradientTape` only watches `tf.Variable`s by default.
          tape.watch(img_rolled)

          # Extract a tile out of the image.
          img_tile = img_rolled[y:y+tile_size, x:x+tile_size]
          loss = calc_loss(img_tile, self.model)
```

(Slika 17) Funkcija TiledGradients()

Za još bolju obradu slike, dobro je iskombinovati podelu slike na proizvoljne pravougaonike i primenu skaliranja, što omogućava kod prikazan na slici 18.

```
In [27]: get_tiled_gradients = TiledGradients(dream_model)
```

```
In [28]: def run_deep_dream_with_octaves(img, steps_per_octave=100, step_size=0.01,
                                         octaves=range(-2,3), octave_scale=1.3):
           base_shape = tf.shape(img)
           img = tf.keras.utils.img_to_array(img)
           img = tf.keras.applications.inception_v3.preprocess_input(img)

           initial_shape = img.shape[:-1]
           img = tf.image.resize(img, initial_shape)
           for octave in octaves:
             # Scale the image based on the octave
             new_size = tf.cast(tf.convert_to_tensor(base_shape[:-1]), tf.float32)*(octave_scale**octave)
             new_size = tf.cast(new_size, tf.int32)
             img = tf.image.resize(img, new_size)

             for step in range(steps_per_octave):
               gradients = get_tiled_gradients(img, new_size)
               img = img + gradients*step_size
               img = tf.clip_by_value(img, -1, 1)

               if step % 10 == 0:
                 display.clear_output(wait=True)
                 show(deprocess(img))
                 print ("Octave {}, Step {}".format(octave, step))

           result = deprocess(img)
           return result
```

(Slika 18) Funkcija run_deep_dream_with_octave()

Konačnu funkciju pozivamo na način prikazan na slici 19, čime dobijamo izlaznu sliku prikazanu na slici 20.

```
In [29]: img = run_deep_dream_with_octaves(img=original_img, step_size=0.01)

         display.clear_output(wait=True)
         img = tf.image.resize(img, base_shape)
         img = tf.image.convert_image_dtype(img/255.0, dtype=tf.uint8)
         show(img)
```

(Slika 19) Poziv konačne funkcije

(Slika 20) Izlazna slika nakon poboljšanja mreže

4. Zaključak

Imitaciju rada DeepDream mreže kompanije Google moguće je postići na različite načine. U zavisnosti od raspoloživih resursa (u vidu memorije, vremena i stanja računara) moguće je implementirati ovu ideju na vrlo jednostavan način, kao što je primenom funkcije run_deep_dream_simple(), ali i primenom mnogo složenijih obrada i izračunavanja koja su u ovom radu prikazana. U zavisnosti od toga na čemu je trenirana mreža koju koristimo, izlaz će se razlikovati čak i nakon istog načina obrade ulaza.
Smatra se da svrha postojanja ovog koncepta nije isključivo zabava, već i pokušaj razumevanja dubljih slojeva neuronskih mreža, kao i samog njenog funkcionisanja, kako bi se buduća rešenja nekih problema manje svodila na isprobavanje različitih modela mreža, a više na dokazanom znanju.

5.Literatura

1. Wikipedia https://www.wikipedia.org/
2. TensorFlow https://www.tensorflow.org/
3. StackOverFlow https://stackoverflow.com/
4. GeeksForGeeks https://www.geeksforgeeks.org/
5. Medium forum https://medium.com/