Математичка гимназија

МАТУРСКИ РАД
из програмирања и програмских језика

Примена дубоког учења на замену лица (*Deepfake*)

Ученик:
Алекса Малков IVб

Ментор:
проф. Снежана Јелић

Београд, мај 2021.

Садржај

Списак слика

Слика 1 је преузета из [*Ima*20], слике 2, 3, 4 из [Ник19], слика 5 из [Du20], слике 6, 7 из [Ngu19], слика 8 из [Hui18].

1 - Увод

Лажирање слика и видео записа није ништа ново, али пре само пар година (2017.) корисник сајта *Reddit* под именом „*deepfakes*“ је објавио алгоритам којим се лако могу направити реални видео снимци неке особе на којима она ради ствари које никада није радила. Тај алгоритам има велики број примена, неких јако лоших али такође и неких добрих, и могао би да има велики утицај на нашу будућност.

Ради се о примени неуронских мрежа, тачније аутоенкодера, који се тренирају да измене слику једне особе тако да изгледа као слика друге особе.

Циљ овог рада је прављење програма за тренирање такве неуронске мреже, као и објашњење ове технологије.

2 - Машинско учење

2.1 Машинско учење и неуронске мреже

Машинско учење је један од најпопуларнијих видова вештачке интелигенције. Користи се у случајевима када покушавамо да решимо проблем који човек не може лако да дефинише или напише алгоритам за његово решавање. Разликује се од обичног програмирања у томе што ми не пишемо сами алгоритам који решава тај проблем, него дефинишемо модел који на основу великог броја података сам „учи“ како да реши проблем.

Пример проблема за који је машинско учење погодно је препознавање лица на сликама. Иако је човеку лако да препозна лице, тешко би могао да дефинише по чему се нека слика лица разликује од осталих слика. Могао би да каже да се лице састоји од очију носа и уста у одређеним односима, али препознавање ових делова скоро је подједнако компликовано. Са друге стране, методи машинског учења попут неуронских мрежа могу да науче да прилично добро препознају лица.

Машинско учење може бити надгледано учење, ненадгледано учење или учење поткрепљивањем. Код надгледаног учења се користи скуп података који садржи парове улазних података и ознаке (податке које би из њих требало добити). Код ненадгледаног учења се користи скуп података без икаквих ознака. Код учења поткрепљивањем постоји агент који у зависности од стања окружења предузима акције и добија награду онда када је ближи остварењу циља.

Најпопуларнија област машинског учења је дубоко учење, које се бави неуронским мрежама. То је тип модела који може да научи да препознаје компликоване особине података. Неуронска мрежа има свој улаз, неколико скривених слојева који се надовезују један на други и један излазни слој. Сваки од слојева има параметре који се користе у рачунању његовог излаза. Тим параметрима се на почетку додељују случајне вредности, а касније се прилагођавају током тренирања мреже.

Први слој неуронске мреже може да научи да препознаје неке особине улазног податка. Затим следећи слој препознаје неке особине излаза првог слоја и тако даље. На тај начин, сваки следећи слој препознаје све комплексније особине почетног податка. Сматра се да је то један од главних разлога успешности неуронских мрежа.

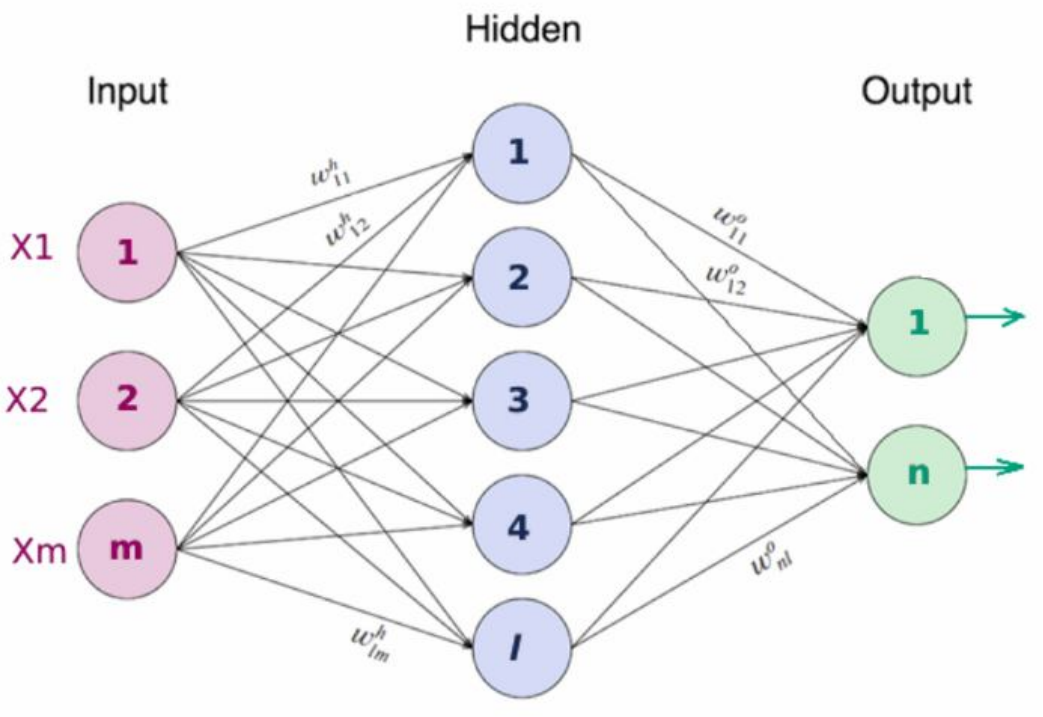
Постоје разни типови неуронских мрежа у зависности од типа слојева. За потребе овог рада најважније су потпуно повезане неуронске мреже и конволутивне неуронске мреже.

2.2 Потпуно повезане неуронске мреже

Потпуно повезане су најједноставнији тип неуронских мрежа. Улаз и излаз једног слоја овакве мреже имају облик једнодимензионих низова. Сваки слој се састоји од неурона. Излаз неурона се рачуна на следећи начин:

$$g\left(\sum_{i=1}^{n} w_i x_i + w_0\right)$$

где су w_i и w_0 параметри тог неурона, који се такође називају и „тежине“ и који се уче током тренирања, x_i су чланови улазног низа, а g је нелинеарна активациона функција.



Слика 1: Потпуно повезана неронска мрежа

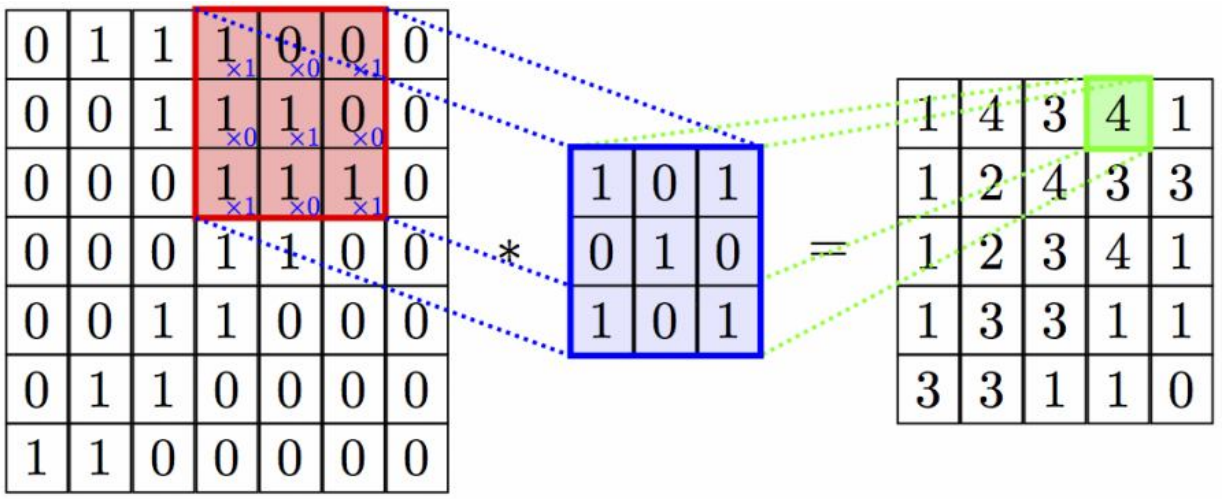
Како је сваком елементу улаза додељена тежина, неурон учи да препозна које особине улазног податка треба да имају већи утицај на излаз неурона. Тај излаз онда постаје особина податка који улази у следећи слој неурона.

Активациона функција је важна јер неурони рачунају линеарну функцију својих улаза, па када не би било активационе функције, или када би она била линеарна, цела мрежа би била композиција линеарних функција. Како је композиција линеарних функција такође линеарна, цела неуронска мрежа би била линеарна па би то било слично као да имамо само један неурон.

2.3 Конволутивне неуронске мреже

Један од проблем код примене потпуно повезаних неуронских мрежа на слике је у томе што не праве никакву везу између пиксела који се налазе једни близу других. Сваки пиксел би био само један елемент улазног низа. То доста отежава проналажење било каквих облика на сликама.

Конволутивне неуронске мреже над улазним податком рачунају операцију конволуције. За одређену дводимензиону матрицу и дводимензиони филтер мањих димензија ова операција се може представити на следећи начин. Замислимо да „померамо“ филтер по матрици као што је приказано на слици.

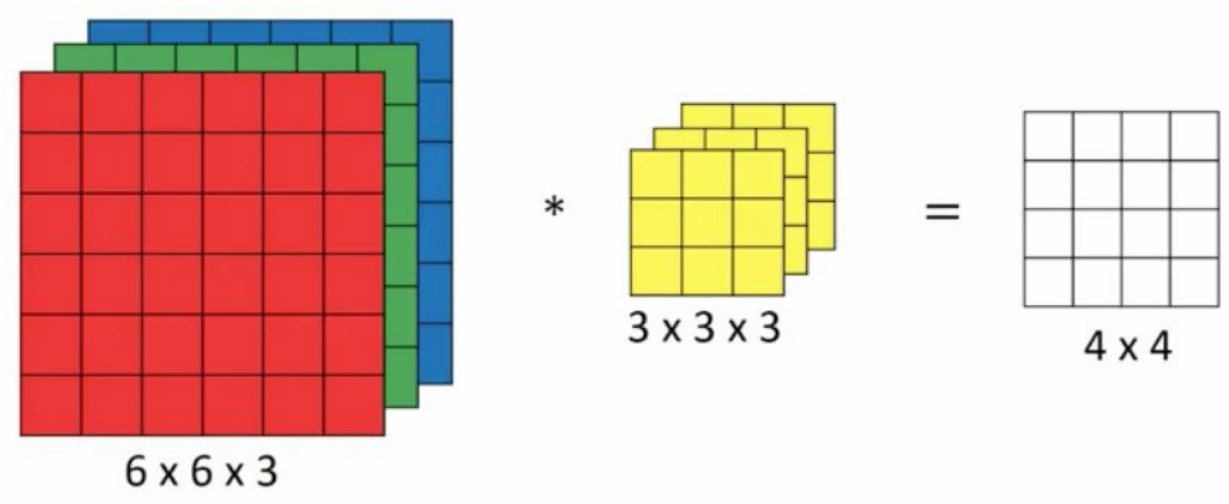


Слика 2: Конволуција улазне слике са филтером

Вредности излазне матрице добијају се тако што издвојимо подматрицу са димензијама филтера, све њене елементе помножимо одговарајућим елементом филтера а затим их саберемо. Након тога, рачуна се активациона функција над елементима излазне матрице. Елементи филтера су параметри конволутивне неуронске мреже.

Видимо да се као резултат добија матрица мањих димензија од почетне. То се може спречити проширивањем почетне матрице пре вршења конволуције, најчешће додавањем нула по ободима. Такође, филтер се не мора померати по један пиксел по почетној матрици већ може имати неки већи корак.

Приликом објашњавања конволуције претпоставили смо да улазна матрица има 2 димензије. Међутим, слике се у компјутерима представљају у три димензије (висина, ширина, број канала). Конволуција над таквим податком вршиће се тако што ће и филтер имати исту трећу димензију као податак, и помераће се по ширини и висини.



Слика 3: Вишеканални филтер

Један слој конволутивне неуронске мреже има више филтера, а број канала излаза једнак је броју филтера.

Операција конволуције може се користити за откривање делова слике са одређеном особином као што су хоризонталне, вертикалне, косе ивице или линије. Филтере у конволутивним мрежама не пишу људи већ оне саме науче да препознају одређене структуре. Ако први слој научи да препознаје ствари попут ивица, следећи ће на основу његових резултата препознати мало комплексније облике, док ће неки од следећих слојева бити у стању да препозна и нешто попут људског лица.

2.4 Тренирање неуронских мрежа

Тренирање неуронске мреже је процес оптимизације њених параметара тако да њен излаз буде сличнији жељеном излазу.

Пре почетка тренирања неопходно је одабрати функцију грешке. То је функција која одређује колико је излаз модела који тренирамо погрешан, то јест

колико се разликује од жељеног. Циљ тренирања је заправо минимизовање функције грешке.

Тренирања се врши понављањем 4 корака:

1. Пролаз у напред – У зависности од типа неуронске мреже, на начин дефинисан у прошлим главама, почевши од неког податка из сета добијамо излаз неуронске мреже.
2. Рачунање функције грешке – Унапред дефинисаном функцијом грешке, упоређујемо излаз мреже и циљну вредност из сета података.
3. Пропагација у назад – Пропагација у назад је алгоритам којим се одређују парцијални изводи функције грешке по параметрима неуронске мреже. Они ће касније бити потребни за оптимизацију. Тачна имплементација овог алгоритма зависи од типа неуронске мреже. О којој год мрежи да се ради, базирана је на коришћењу правила за одређивање парцијалног извода сложене функције.

$$z = f(y), y = g(x)$$
$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$$

4. Оптимизација – Параметри се у сваком кораку тренирања постепено прилагођавају са циљем да се дође ближе минимуму функције грешке. Најједноставнија метода оптимизације је градијентни спуст. Том методом се сваки параметар у сваком кораку мења на следећи начин:

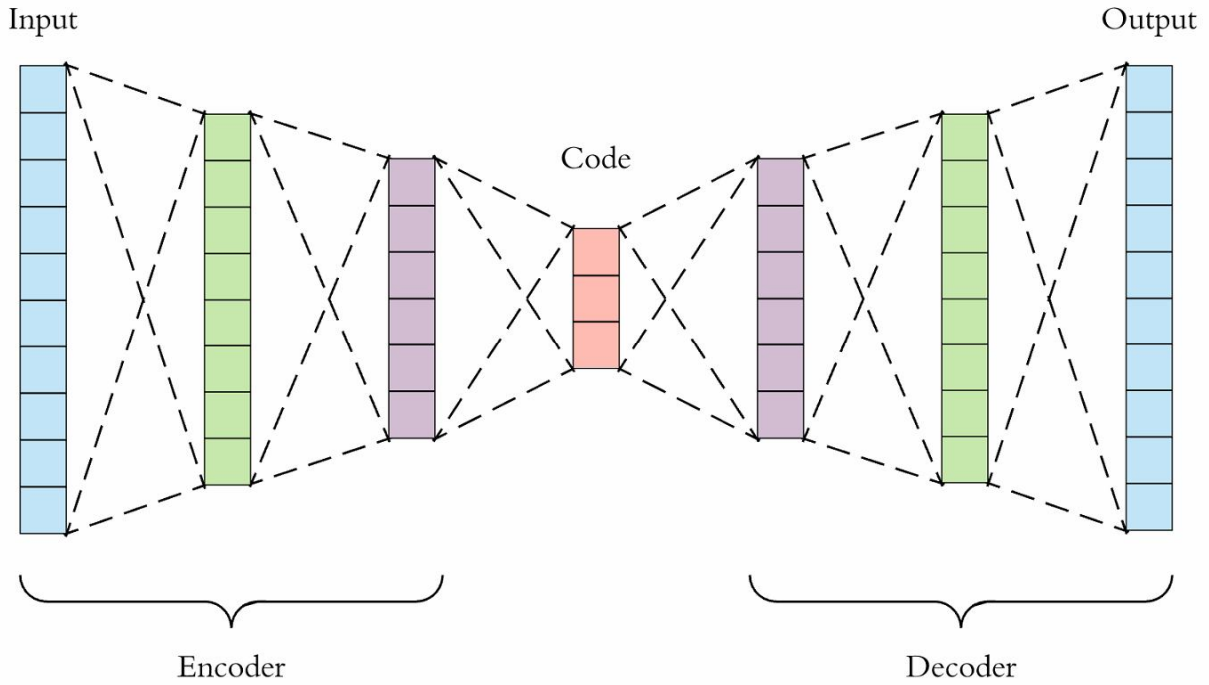
$$w = w - \alpha\frac{\partial f}{\partial w}$$

где је f функција грешке, а параметар који мењамо. Параметар α се назива „брзина учења“ (енгл. *learning rate*) и одређује колико мењамо параметре модела у сваком кораку.

Постоје и разне друге методе које се могу користити уместо градијентног спуста. За тренирање неуронских мрежа најчешће се користи метода Адам, која осим тренутног парцијалног извода такође укључује и оне који су употребљени у прошлим корацима, као и њихове квадрате. То омогућује већу доследност приликом приближавања минимуму функције.

2.5 Аутоенкодер

Аутоенкодер је неуронска мрежа која има задатак да реконструише улазни податак. Састављена је од два дела: енкодера и декодера. Енкодер као излаз има податак знатно мањих димензија од почетних, док декодер из тога покушава да реконструише почетни податак.



Слика 4: Аутоенкодер

Вредност овакве неуронске мреже је у томе што као излаз енкодера добијамо податак мањих димензија који садржи све информације које су потребне да почетни податак буде реконструисан. На тај начин се могу издвојити битне карактеристике почетног податка.

Аутоенкодер не мора увек враћати исти податак који добија као свој улаз. Пример тога је аутоенкодер за уклањање шума. Он као улаз добија слику са додатим шумом, енкодер је умањује и издваја само битне карактеристике, довољне да декодер реконструише оригиналну слику (без шума).

2.6 Конволутивне неуронске мреже у аутоенкодеру

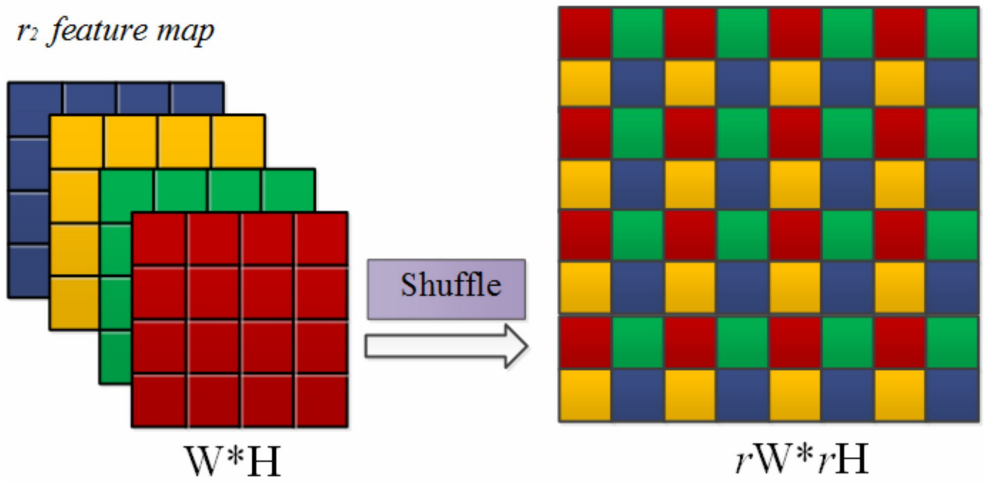
Ако бисмо желели да користимо аутоенкодер за обраду слика, пожељно би било користити конволутивне неуронске мреже. Видели смо да се у аутоенкодеру димензије податка прво смањују па после повећавају. Постоји неколико начина да се то изведе применом конволутивних неуронских мрежа.

Излаз слоја конволутивне неуронске мреже има три димензије: висину, ширину и број канала. Број канала излазног податка једнак је броју филтера тог слоја, тако да је проблем мењања величине податка смањен на висину и ширину.

Током вршења конволуције, филтер се не мора померати за само по један пиксел. Најједноставнији начин да смањимо дужину и ширину излазног податка је да повећамо тај корак. Ако бисмо на пример користили корак величине 2, излаз би био 4 пута мањи. Енкодер се може направити понављањем таквог конвлутивног слоја неколико пута.

Један од начина да се висина и ширина повећају је коришћење операције *pixel shuffle*. То је операција која преобликује тензор тако да има мањи број канала, а већу висину и ширину.

Ова операција има један параметар, величину блока (r), и мења облик податка из тројке (C, H, W) у тројку (C/r^2, H x r, W x r), где је C број канала, H висина и W ширина, тако што ствара блокове димензија r x r у које се смештају бројеви са истим координатама у различитим каналима, као што је приказано на слици (Слика 5).



Слика 5: Операција *pixel shuffle*

Видимо да овде нема никаквих параметара који се уче током тренирања мреже. Ова операција се користи у комбинацији са једним слојем конволутивне неуронске мреже. Понављањем те комбинације неколико пута може се направити декодер.

2.7 Аугментација података

Тренирање неуронских мрежа захтева јако велику количину података. Уколико се тренирање врши предуго на недовољном броју података, долази до проблема који се назива „претерано учење“ (енгл. *overfitting*). То значи да модел

научи да даје задовољавајуће резултате за улазне податке из сета који се користи за тренирање, али не и ако се користи било који други податак.

Како није увек лако повећати скуп података, он може бити „вештачки повећан“ применом аугментације података. То подразумева вршење случајних промена над подацима непосредно пре корака тренирања. На пример, над сликама се могу вршити трансформације попут ротације, транслације, осне симетрије, мењања висине и ширине слике... Тако ће иста слика сваки пут када буде употребљена за тренирање изгледати мало другачије и створити сличан ефекат као да је број слика у скупу за учење већи.

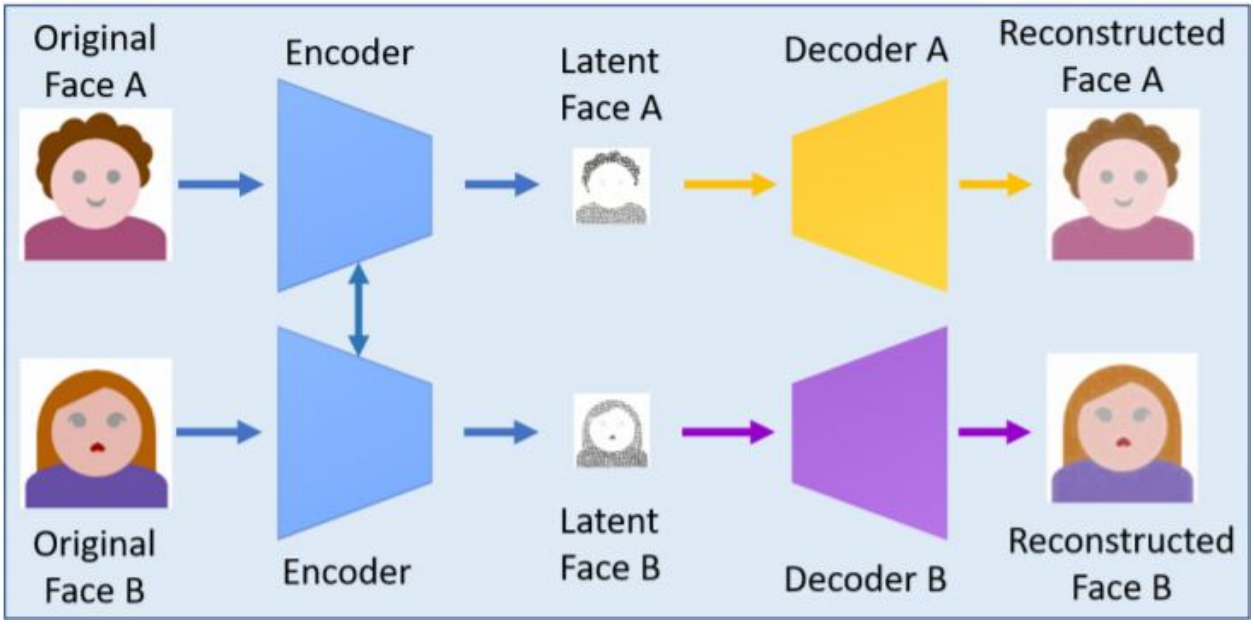
Приликом аугментације битно је пазити да податак не променимо превише како би и даље био користан за учење. Престаће да буде користан онда када не изгледа као подаци на којима ћемо касније користити трениран модел.

У случају овог програма, ми ћемо модел користити на сликама на којима је лице вертикално или скоро вертикално и центар лица се налази скоро у центру слике. Ако бисмо слику превише заротирали или транслирали, то би само отежало учење, док мале промене могу да буду корисне. Слично важи и за мењање ширине и висине слике. Осна симетрија може бити корисна само по вертикалној оси (под претпоставком да су лица симетрична).

Аугментација података који ће се користити у следећем кораку тренирања може се вршити паралелно са тренирањем модела.

3 - Алгоритам *Deepfake*

Алгоритам *Deepfake* се састоји из тренирања неуронске мреже (аутоенкодера), која као улаз има уско исечену слику лица једне особе (особе А), а као излаз, лажну, уско исечену слику лица друге особе (особе Б) са истим положајем и изразом лица. Та мрежа може бити употребљена на више начина. Може се користити за замену лица на фотографији тако што се лице исече са слике особе А, искористи као улаз у мрежу и замени лицем особе Б добијеним као излаз мреже. Такође може бити употребљена за замену израза лица или положаја усана на слици особе Б. За то би било потребна слика особе А са истим положајем лица, са које би лице било исечено и употребљено као улаз неуронске мреже. Лажни видео може бити направљен понављањем неког од ових поступака на сваком фрејму.

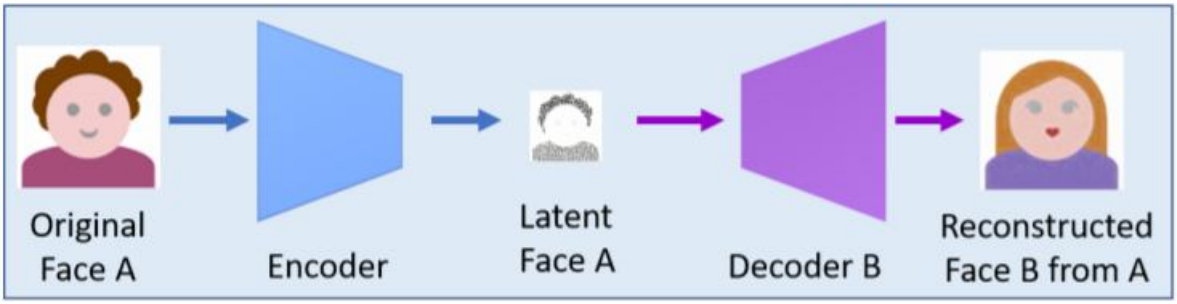


Слика 6: Тренирање аутоенкодера за *Deepfake*

Први корак у прављењу „*deepfake*“ садржаја је сакупљање података. Потребна су 2 скупа података који садрже бар по неколико стотина слика лица по једне особе.

Следећи корак је тренирање неуронских мрежа. Паралелно се тренирају два аутоенкодера са заједничким енкодером али различитим декодерима. Сваки од њих има задатак да реконструише слике једне особе (аутоенкодер А као улаз има слику особе А, а као излаз треба да добије исту ту слику). Интуиција иза овога је да ће се у излазу енкодера задржати информације битне за оба лица (као што су положај и израз лица), а које су довољне да декодери из тога реконструишу лице за које су тренирани (Слика 6).

Након тренирања, слике особе А можемо „претворити“ у слике особе Б тако што бисмо их користили као улаз у аутоенкодер Б (Слика 7).



Слика 7: Примена аутоенкодера за *Deepfake*

Једна од предности овог алгоритма је то што није потребно да подаци имају било какве ознаке, нити је потребно да упарујемо слике једне и друге особе. Просто нам требају 2 потпуно независна сета података који садрже само слике. Дакле ради се о ненадгледаном учењу. Такве сетове података је много лакше направити. На пример, можемо пронаћи неколико видео снимака ових особа и издвојити слике из њих.

Испоставља се да се добијају бољи резултати ако се као улаз током тренирања користе изобличене слике. Интуиција иза овога је да аутоенкодер А треба да научи да реконструише лице особе А чак и ако улазна слика нема исти облик и карактеристике лица.

4 - Имплементација

4.1 Модел

За имплементацију сам користио програмски језик *Python* и библиотеку за дубоко учење *TensorFlow*.

Конструисање неуронских мрежа са одговарајућим хипер параметрима је тежак посао који захтева много експериментисања, поготово код задатака као што је овај, који захтевају релативно велику и компликовану мрежу. У великом броју случајева је исплативије користити моделе које су други направили и који су већ дали добре резултате. Зато сам одлучио да за овај рад користим неуронску мрежу са истим хипер параметрима као она која је коришћена у оригиналном коду за deepfake. Након изласка оригиналног кода 2017. до данас направљени су и неки комплекснији модели који су давали боље резултате али ја ћу користити оригинални модел јер је за његово тренирање потребно мање времена. Улаз и излаз модела су слике са 64*64 пиксела.

Прво се дефинишу блокови за повећање и смањење података у аутоенкодеру.

```
def downscale(filters, x):
    return layers.Conv2D(filters, (5, 5), strides=2, padding='same',
                         activation=tf.nn.leaky_relu)(x)

def upscale(filters, x):
    x = layers.Conv2D(filters, (3, 3), padding='same',
                      activation=tf.nn.leaky_relu)(x)
    return layers.Lambda(lambda l: tf.nn.depth_to_space(l, 2))(x)
```

Као што је објашњено раније, за смањење података користи се конволутивна мрежа са кораком величине 2, а за повећање конволутивна мрежа у комбинацији са операцијом *pixel shuffle* (која је у библиотеци *TensorFlow* имплементирана као функција `nn.depth_to_space`). Слојеви су имплементирани у оквиру модула

tensorflow.keras.layers. Од функције depth_to_space је потребно да се направи одговарајући ламбда слој који је извршава.

```
def new_encoder():
    input = layers.Input(shape=(64, 64, 3))
    x = downscale(128, input)
    x = downscale(256, x)
    x = downscale(512, x)
    x = downscale(1024, x)
    x = layers.Flatten()(x)
    x = layers.Dense(1024)(x)
    x = layers.Dense(4 * 4 * 1024)(x)
    x = layers.Reshape((4, 4, 1024))(x)
    x = upscale(2048, x)
    return Model(inputs=input, outputs=x, name="encoder")

def new_decoder():
    input = layers.Input(shape=new_encoder().output_shape[1:])
    x = upscale(1024, input)
    x = upscale(512, x)
    x = upscale(256, x)
    x = layers.Conv2D(3, (5, 5), padding='same', activation='sigmoid')(x)
    return Model(inputs=input, outputs=x, name="decoder")

def autoencoders():
    encoder = new_encoder()
    decoder_a = new_decoder()
    decoder_b = new_decoder()

    input = layers.Input(shape=(64, 64, 3))
    x = encoder(input)
    a = decoder_a(x)
    b = decoder_b(x)

    autoencoder_a = Model(inputs=input, outputs=a)
    autoencoder_b = Model(inputs=input, outputs=b)
    return autoencoder_a, autoencoder_b
```

Затим правим функције за прављење енкодера и декодера као и функцију за прављење два аутоенкодера који имају заједнички енкодер. Она је касније у коду употребљена за дефинисање модела.

```
autoencoder_a, autoencoder_b = autoencoders()
autoencoder_a.summary()
learning_rate = 5e-5
optimizer = tf.keras.optimizers.Adam(
    lr=learning_rate, beta_1=0.5, beta_2=0.999)
```

```
loss = tf.keras.losses.MeanAbsoluteError()
autoencoder_a.compile(optimizer, loss)
autoencoder_b.compile(optimizer, loss)
```

За тренирање модела користи се оптимизатор Адам са истим параметрима као у оригиналној имплементацији. Као функција грешке користи се средња апсолутна грешка.

4.2 Припрема података

Сет података који сам користио током тестирања овог кода је сет са сликама Доналда Трампа и Николаса Кејџа, који је објављен заједно са оригиналном имплементацијом *deepfake*-а и од тада се користи за тестирање различитих модела [*Wu*17]. Овај код може бити коришћен и за тренирање на било ком другом сету у ком су слике исечене на сличан начин (тако да је сличан однос величине слике и лица на њој).

Да би били употребљени, подаци се прво морају прочитати са диска.

```
def read(file_path):
    img = tf.io.read_file(file_path)
    img = tf.image.decode_jpeg(img, channels=3)
    img = tf.image.resize(img, (128,128))
    img /= 255.
    return img
```

Ова функција чита податке и мења им величину.

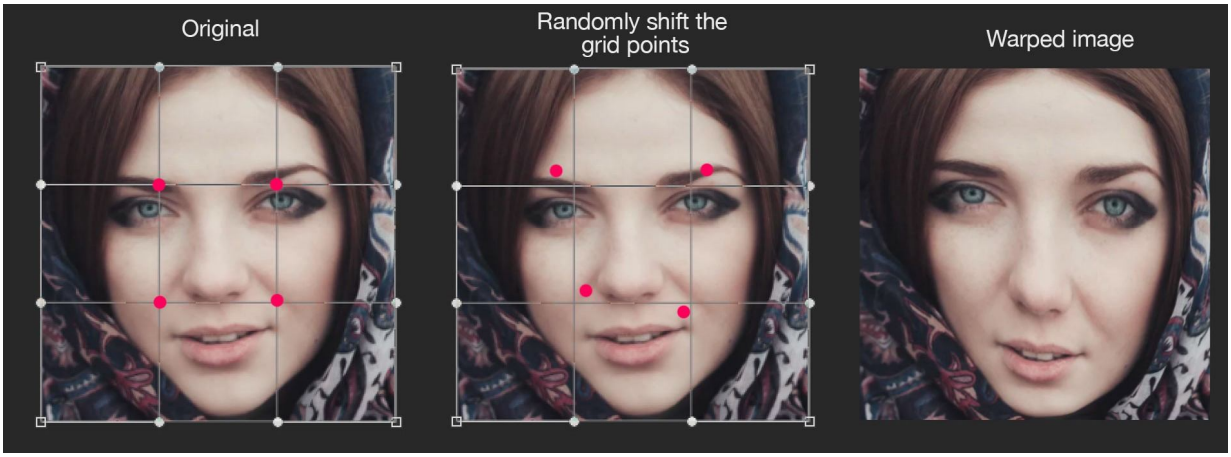
Аугментација и изобличавање ће се вршити на сликама величине 128*128 пиксела, након чега ће оне бити исечене и промењене у величину 64*64.

```
data_augmentation = tf.keras.Sequential(
    [
        layers.experimental.preprocessing.RandomFlip("horizontal"),
        layers.experimental.preprocessing.RandomWidth(0.05),
        layers.experimental.preprocessing.RandomHeight(0.05),
        layers.experimental.preprocessing.RandomRotation(0.03),
        layers.experimental.preprocessing.RandomTranslation(0.03, 0.03)
    ]
)

crop = tf.keras.Sequential(
    [
        layers.experimental.preprocessing.CenterCrop(80, 80),
        layers.experimental.preprocessing.Resizing(64, 64)
    ]
)
```

Аугментација и сечење имплементирани су коришћењем *Keras* слојева, слично као и неуронска мрежа.

```
mapx = np.broadcast_to(np.linspace(64 - 40, 64 + 40, 5),
                       (5, 5)).astype('float32')
mapy = mapx.T.astype('float32')
dest0 = np.vstack((mapy.reshape((25)),
                   mapx.reshape((25)))).T.astype('float32')
dest0 = tf.convert_to_tensor(dest0)

def random_warp(img):
    size = tf.shape(img)[0]
    dest = tf.broadcast_to(dest0, (size, 25, 2))
    source = dest + tf.random.normal(shape=(1, 25, 2), stddev=2.5)
    warped_images, _ = tfa.image.sparse_image_warp(img, source, dest)
    return warped_images, img
```

Слике се изобличују коришћењем функције `sparse_image_warp` из библиотеке `tensorflow_addons`. Она као улаз прима слику, низ са координатама тачака и низ са координатама на које те тачке треба преместити и на основу тога изобличује слику (Слика 8).



Слика 8: Изобличавање слике

Како тренирање овог модела може да потраје јако дуго, важно је потрудити се да се програм извршава што брже. Библиотека *Tensorflow* омогућава да се припрема података извршава паралелно са тренирањем. Дефинишем објекат `tensorflow.data.Dataset` а затим дефинишем функције које се над њим требају извршити.

```
AUTOTUNE = tf.data.AUTOTUNE

def prepare_data(data_dir, augment=True, warp=True):
    ds = tf.data.Dataset.list_files(data_dir + "/*")
    ds = ds.shuffle(1000)
    ds = ds.map(read,
                num_parallel_calls=AUTOTUNE)
    ds = ds.batch(batch_size)
    ds = ds.cache()

    if augment:
        ds = ds.map(lambda x: (data_augmentation(x, training=True)),
                    num_parallel_calls=AUTOTUNE)

    if warp:
        ds = ds.map(random_warp)
        ds = ds.map(lambda x, y: (crop(x, training=True),
                                  crop(y, training=True)),
                    num_parallel_calls=AUTOTUNE)
    else:
        ds = ds.map(lambda x: (crop(x, training=True),
                               crop(x, training=True)),
                    num_parallel_calls=AUTOTUNE)

    ds = ds.repeat()
    ds = ds.prefetch(buffer_size=AUTOTUNE)
    return ds
```

Након читања података и распоређивања у тренинг пакете (т.ј. скупове података који се тренирају паралелно), позива се функција `cache` која омогућава чување припремљених података у оперативној меморији. На тај начин неће бити неопходно поновно читање податка у свакој епохи, што убрзава извршавање програма.

Након тога се врше аугментација, изобличавање и исецање слика. На крају се добијају 2 слике, једна изобличена која ће се користити као улаз у мрежу и једна нормална која ће бити циљ за излаз. Ако не желимо да изобличавамо слике добиће се две исте слике. На крају се позива функција prefetch која омогућава припремање следећег бача података, паралелно са тренирањем. Аргумент `AUTOTUNE` код неких функција значи да програм сам одлучује у тренутку извршавања у колико би паралелних процеса било најбоље извршити функцију, или у случају prefetch, колико података припремити унапред.

Касније у коду користим ову функцију за дефинисање сетова података.

```
ds_a = prepare_data(data_dir_a, augment=True, warp=True)
ds_b = prepare_data(data_dir_b, augment=True, warp=True)
```

4.3 Чување научене мреже

Након завршетка, као и током тренирања модела, важно је повремено сачувати параметре, за случај да се тренирање прекине или почне да даје лоше резултате. Функције за чување и учитавање параметара изгледају овако:

```
def save_weights(num=0):
    autoencoder_a.get_layer("encoder").save_weights(
        "weights/encoder" + str(num))
    autoencoder_a.get_layer("decoder").save_weights(
        "weights/decoder_a" + str(num))
    autoencoder_b.get_layer("decoder").save_weights(
        "weights/decoder_b" + str(num))
    print("Weights " + str(num) + " saved.")

def load_weights(num=0):
    autoencoder_a.get_layer("encoder").load_weights(
        "weights/encoder" + str(num))
    autoencoder_a.get_layer("decoder").load_weights(
        "weights/decoder_a" + str(num))
    autoencoder_b.get_layer("decoder").load_weights(
        "weights/decoder_b" + str(num))
    print("Weights " + str(num) + " loaded.")
```

4.4 Приказивање слика

Током тренирања, потребно је пратити какви су тренутни резултати. У овом случају, тешко је закључити колико је модел добар само на основу функције грешке. Зато сам направио функцију која генерише слике из којих се може видети колико је модел тренутно добар.

```
def create_preview_batches():
    set_a = sorted(os.listdir(data_dir_a))[-4:]
    set_b = sorted(os.listdir(data_dir_b))[-4:]
    batch_a = []
    batch_b = []
    for i in range(4):
        batch_a.append(
            crop(np.expand_dims(read(data_dir_a + '/' + set_a[i]), 0))[0])
        batch_b.append(
            crop(np.expand_dims(read(data_dir_b + '/' + set_b[i]), 0))[0])
    return batch_a, batch_b
```

```
def save_preview(name):
    images_a, images_b = create_preview_batches()
    savepath = "previews/" + str(name) + ".png"
    plt.figure(figsize=(12, 8))

    for i in range(4):
        plt.subplot(4, 6, 6*i + 1)
        plt.imshow(images_a[i])
        plt.axis("off")

        plt.subplot(4, 6, 6*i + 2)
        plt.imshow(autoencoder_a(np.expand_dims(images_a[i], 0))[0])
        plt.axis("off")

        plt.subplot(4, 6, 6*i + 3)
        plt.imshow(autoencoder_b(np.expand_dims(images_a[i], 0))[0])
        plt.axis("off")

        plt.subplot(4, 6, 6*i + 4)
        plt.imshow(images_b[i])
        plt.axis("off")

        plt.subplot(4, 6, 6*i + 5)
        plt.imshow(autoencoder_b(np.expand_dims(images_b[i], 0))[0])
        plt.axis("off")

        plt.subplot(4, 6, 6*i + 6)
        plt.imshow(autoencoder_a(np.expand_dims(images_b[i], 0))[0])
        plt.axis("off")
    plt.subplots_adjust(hspace=0.02, wspace=0.02,
                        left=0, right=1, top=1, bottom=0)
    plt.savefig(savepath, bbox_inches='tight', pad_inches=0.04)
    print('Preview saved.')
```

Узимају се увек исте слике из скупа података и користе као улаз за оба аутоенкодера. Дакле добијају се реконструкције тих слика, као и слике другог лица које се из њих добијају. Пример слике приказане овим кодом је наведен у резултатима (Слика 9).

4.5 Тренирање

Функција која тренира модел изгледа овако:

```
def train(steps=20000, save_interval=2500, preview_interval=2500,
          print_interval=250, backup_interval=10000):
    print("Training:")
```

```
iter_a = iter(ds_a)
iter_b = iter(ds_b)
loss_a = []
loss_b = []
for i in range(1, steps + 1):

    batch_a, target_a = next(iter_a)
    loss_a.append(autoencoder_a.train_on_batch(batch_a, target_a))

    batch_b, target_b = next(iter_b)
    loss_b.append(autoencoder_b.train_on_batch(batch_b, target_b))

    if (i % print_interval == 0):
        loss_a_avg = sum(loss_a) / len(loss_a)
        loss_b_avg = sum(loss_b) / len(loss_b)
        loss_a = []
        loss_b = []
        print(i, loss_a_avg, loss_b_avg)
    if(i % save_interval == 0):
        save_weights(0)
    if(i % backup_interval == 0):
        save_weights(i)
    if(i % preview_interval == 0):
        save_preview(i)

print("Training finished.")
```

Дефинишу се итератори кроз сет података, а затим се у петљи врши тренирање модела на датим подацима. После одређеног броја корака, исписује се просечна вредност функције грешке. Такође се врши чување модела и генерисање слика. Са индексом 0 се чува најновији модел, док се резервне копије чувају ређе, са индексом једнаким броју тренутног корака.

5 - Резултати и дискусија

5.1 Резултати

Представљени модел је трениран кроз 65.000 корака. Већина слика испада јако добро након толико тренирања (Слика 9). Нешто лошији резултати се могу видети на сликама са лошијим осветљењем, на којима је лице окренуто или је израз лица необичан.

Важно је нагласити да је тренирање вршено на релативно малом скупу података, који није направљен тако да модел испадне што боље, већ напротив за тестирање различитих модела. Има простора за даља побољшавања резултата, путем додатног експериментисања са променом брзине учења (енгл. *learning rate*) или искључивањем изобличавања пред крај тренинга.

Као што је већ објашњено у опису алгоритма, он може да се примени и само на део слике. Једноставан код за коришћење трениранога модела за замену лица на делу слике изгледа овако:

```
image = read('data/trump/477320164.jpg')
image2 = image[24:104,24:104,:]
image2 = tf.image.resize(image2, (64,64))
image2 = autoencoder_a(np.expand_dims(image2, 0))[0]
image2 = tf.image.resize(image2, (80,80))
image = image.numpy()
image[24:104,24:104,:] = image2
plt.imshow(image)
plt.axis("off")
plt.savefig("slika2.png", bbox_inches='tight')
```

Његовом применом се од слике (Слика 10а) добија слика (Слика 10б).

Слика 9: Резултати након 65.000 корака тренирања
По колонама: (1) оригиналне слике Николаса Кејџа; (2) слике (1) реконструисане применом аутоенкодера; (3) слике (1) добијене замењеним декодером; (4) оригиналне слике Доналда Трампа; (5) слике (4) реконструисане применом аутоенкодера; (6) слике (4) добијене замењеним декодером.

Слика 10: Резултати замене лица
(а) Лево је оригинална слика. (б) Десно је слика са замењеним ликом.

5.2 Могућа унапређења

Неке од других имплементација овог алгоритма садрже бројна побољшања. Нека од њих су:

- бољи модели;
- друге функције грешке - добри резултати су постизани коришћењем индекса структурне сличности (*SSIM*);
- коришћење маски тако да се тренирање не врши коришћењем целих слика, него само делова где је лице;
- повећање вредности функције грешке која се добија на важнијим деловима слике (у областима очију или уста) како би модел пре научио да их генерише како треба;
- додавање филтера за обраду слика који омогућавају неприметније замењивање лица.

Овај рад је фокусиран на прављење програма за тренинг неуронске мреже. Ипак, програм би био потпунији ако би могао да се користи и за прављење скупа података као и коришћене трениране мреже за замену лица на видео снимцима.

5.3 Примена

Од почетка је јасно да ова технологија може бити јако опасна јер је настала како би се користила за прављење лажног садржаја. Тај садржај се може користити за разне врсте превара. На пример могу се направити лажни снимци политичара како причају ствари које им не иду у прилог, за које би просечна особа могла врло лако помислити да су прави. Такође је коришћен и за финансијске преваре и прављење порнографског садржаја.

Развој ове технологије ствара опасност да ћемо ретко моћи да будемо сигурни да је снимак који видимо прави.

Ипак, треба напоменути да постоји и низ позитивних примена. На пример, ова технологија је већ почела да се примењује у филмској индустрији за специјалне ефекте. Може бити употребљена за подмлађивање глумаца, појављивање глумаца у сценама које нису могли да одглуме физички јер су били спречени или појављивање глумаца у опасним сценама. Такође се може користити за превођење снимака на друге језике (тако да изгледа као да особа на снимку заиста отвара уста као да говори други језик иако га заправо не зна). Може се користити у рекламне сврхе. Познате личности могу рекламирати производе без потребе да физички

глуме у тим рекламама. За сада, најчешћа примена је била прављење забавних снимака на интернету.

Занимљиво је да се сличан приступ може применити и на друге врсте података. Постоје програми који могу изменити глас особе тако да звучи као да припада некој другој особи.

6 - Закључак

У овом раду је представљен алгоритам „*deepfake*“ за замену лица, као и основни концепти машинског учења потребни за његову имплементацију. Највише пажње је посвећено неуронским мрежама, аутоенкодерима, као и самом алгоритму њиховог тренирања и примени на овај проблем.

Сматрам ову област јако занимљивом и драго ми је што сам се опробао у прављењу практичног рада који је примењује. Током израде, научио сам много тога новог о овим областима ако и о библиотеци *TensorFlow* коју сам користио за имплементацију. Ово искуство је додатно проширило моје интересовање за ову област и свакако да ћу наставити да учим о њој.

Надам се да ће овај рад помоћи другима који су заинтересованим за ову тему.

Литература

[*CurNPN*]
Nico Curti, **Pixel Shuffle Layer**,
https://nico-curti.github.io/NumPyNet/NumPyNet/layers/pixelshuffle_layer.html

[*Du*20]
Juan Du, Huixin Zhou, Kun Qian, Wei Tan, Zhe Zhang, Lin Gu, Yue Yu, **RGB-IR Cross Input and Sub-Pixel Upsampling Network for Infrared Image Super-Resolution**, *Sensors* 2020, 20, 281.
https://doi.org/10.3390/s20010281

[*Hui*18]
Jonathan Hui, **How deep learning fakes videos (Deepfake) and how to detect it?** *www*, 2018.
https://jonathan-hui.medium.com/how-deep-learning-fakes-videos-deepfakes-and-how-to-detect-it-c0b50fbf7cb9

[*Ima*20]
Ahmed Imam, **Neural Networks**, *www*, 2020.
https://medium.com/swlh/neural-networks-4b6f719f9d75

[*Ngu*19]
Thanh Thi Nguyen, Quoc Viet Hung Nguyen, Cuong M. Nguyen, Dung Nguyen, Duc Thanh Nguyen, Saeid Nahavandi, **Deep Learning for Deepfakes Creation and Detection: A Survey**, *ArXiv.org*, 2019.

[Ник19]
Младен Николић, Анђелка Зечевић, **Машинско учење**, скрипта, 2019.

[*TFdoc*]
Better performance with the tf.data API, TensorFlow, official docs,
https://www.tensorflow.org/guide/data_performance

[*Tor*19]
torzdf, **Training in Faceswap**, *faceswap*, 2019,
https://forum.faceswap.dev/viewtopic.php?t=146

[*Wu*17]
Joshua Wu, **deepfakes_faceswap**, *GitHub*, 2017,
https://github.com/joshua-wu/deepfakes_faceswap

[*Wiki*]
Deepfake, *Wikipedia*, https://en.wikipedia.org/wiki/Deepfake