Univerzitet u Nišu

**Elektronski fakultet**

**Izveštaj istraživačkog projekta iz dubokog učenja**

**– segmentacija vatre iz slika**

Predmet: Duboko učenje

Niš, 2023.

Mentor: Prof. dr. Aleksandar Milosavljević Studenti: Filip Milisavljević, Andrija Malbaša

**Sadržaj**

[1. Opis problema 1](#_Toc169038978)

[2. Opšti opis projekta 1](#_Toc169038979)

[3. Glavne komponente 2](#_Toc169038980)

[3.1 *segmentation\_models* bilioteka: 3](#_Toc169038981)

[3.2 Albumentations biblioteka 3](#_Toc169038982)

[3.3 Tensorflow i Keras 4](#_Toc169038983)

[3.4 OpenCV 5](#_Toc169038984)

[4. Implementacija 5](#_Toc169038985)

[4.1 Priprema podataka: 6](#_Toc169038986)

[4.2 Konfiguracija projekta 7](#_Toc169038987)

[4.3 Treniranje 7](#_Toc169038988)

[4.4 Preliminarna evaluacija 9](#_Toc169038989)

[4.5 Konačna evaluacija 9](#_Toc169038990)

[5. Rezultati 10](#_Toc169038991)

[6. Zaključak 16](#_Toc169038992)

[7. Reference 17](#_Toc169038993)

# Opis problema

U radu pod nazivom „Fire Detection and Localization Method Based on Deep Learning in Video Surveillance [1.]“ je tema detekcija i lokalizacija vatre na slikama. Detekcija spada u trivijalne probleme dubokog učenja jer se svodi na klasifikaciju slika na one s vatrom u njima i one bez, i , stoga, nije tema ovog projekta. Lokalizacija se u naučnom radu radi deljenjem slika na superpiksele pomoću SLIC algoritma, nakon čega se svaki od njih posle „klasifikuje“ pomoću CNN arhitekture na superpiksel sa vatrom i bez vatre. Od detektovanih superpiksela se posle pravi segmentaciona maska.

U radu opisuju okvir za detekciju i lokalizaciju požara zasnovan na dubokom učenju. Tradicionalne metode detekcije i lokalizacije požara imaju sporu brzinu detekcije, nisku preciznost detekcije i nisku preciznost lokalizacije, pa su istraženi modele dubokog učenja kako bi se ovi problemi rešili. Prvo izdvajaju sličice iz video nadzora. Zatim primenjuju CNN model dubokog učenja da bi detektovali požar na izdvojenim ključnim sličicama videa. Na kraju je lokalizacija požara pomoću superpiksela i CNN-a na ključnim sličicama videa u kojima je izbio požar.

Cilj ovog projekta je uporediti rezultate različitih varijanti U-Net arhitektura sa onima iz ovog rada, sa manje podataka neophodnih za efektivno treniranje svake od njih.

# Opšti opis projekta

Skup podataka sa kojim se radi u navedenom radu (fire-dunnings-dataset) sadrži 224x224 slike, zajedno sa njihovim superpikselima, koji spadaju u jednu od dve kategorije: fire i nofire. Iz tog skupa podataka će se razmatrati isključivo podskup za evaluaciju tj. testiranje UNET modela.

Za treniranje U-Net arhitektura je korišćen skup podataka pod nazivom „Fire and Smoke Segmentation Image Dataset“ koji sadrži slike zajedno sa njihovim segmentacionim maskama koji prikazuju lokaciju vatre unutar slike. Podskup je izdeljen na podskupove za trening, validaciju i testiranje.

Isprobane su UNET arhitekture sa „backbone“ enkoderima:

* vgg19,
* resnet101
* mobilenetv2
* inceptionv3

, i funkcijama gubitka:

* Binary Cross-entropy (BCE)
* Dice
* Jaccard
* kombinacija BCE i Dice
* kombinacija BCE i Jaccard

i metrikom:

* Intersection over Union

Tako trenirani modeli se potom preliminarno evaluiraju nad test podskupom skupa podataka Fire and Smoke Segmentation Image Dataset-a na osnovu Intersection over Union metrike. Najbolji modeli se potom isprobavaju nad test skupom podataka iz fire-dunnings-dataset-a, i sakupljaju se zabeleženi rezultati u vidu: True Positive Rate (TPR), False Positive Rate (FPR), Precision, Recall, Accuracy, **na nivou svih superpiksela u skupu podataka, a ne na nivou svake slike ponaosob.**

**Korišćene metrike:**

**,**

**,**

**,**

**,**

**,**

***.***

# Glavne komponente

## *segmentation\_models* bilioteka:

*segmentation\_models* [2] je Python paket otvorenog koda koji sadrži implementacije popularnih arhitektura za semantičku segmentaciju slika. Glavne pogodnosti ove bilioteke su:

* API visokog nivoa – model se može kreirati i kompajlirati sa dve linije koda, pri čemu je moguće definisati kompletnu arhitekturu za potrebe ovog projekta u samo jednoj (backbone mreža za U-Net, aktivaciona fukncija, broj klasa za segmentaciju, indikatore za zamrzavanje slojeva tokom treniranja i fine-tuninga...)
* Backbone mreže sa pretreniranim težinama za bržu i bolju konvergenciju tokom treninga
* Popularne metrike za semantičku segmentaciju:
  + Intersection over Union
  + Precision
  + Recall ...
* Popularne funkcije gubitaka za semantičku segmentaciju:
  + Jacard
  + Dice
  + Binary Cross-Entropy
  + Categorical Cross-Entropy ...
* Arhitekture 4 modela za binarnu i višeklasnu segmentaciju slika:
  + PSPnet
  + Linknet
  + FPN
  + U-Net

Biblioteka je napravljena tako da radi sa *Keras i Tensorflow Keras* framework-ovima (podrazumavno traži *Keras* na sistemu, a ako ga ne nađe, pokušaće sa *tf.keras*). Moguće je specificirati sa kojim formatom slika se radi (broj kanala na prvom ili poslednjem mestu u shape-u slike). Kreirani modeli za segmentaciju su instance Keras Model klase i u slučaju U-Net arhitekture je moguće odabrati backbone mrežu (sa ili bez težina prenaučenih na ImageNet skupu podataka).

## Albumentations biblioteka

Albumentations [3] je Python bilioteka otvorenog koda za augmentaciju slika. Nudi fleksibilan skup transformacija koje se mogu primeniti na slike ali i na njihove maske istovremeno, što ih čini pogodnim za potrebe projekta semantičke segmentacije.

Za potrebe semantičke segmentacije slika, neophodno je istovremeno primeniti augmentaciju i na sliku i na njegove maske (može ih biti više, zavisno od broja klasa). Albumentations bilioteka osigurava da i na ulaznu sliku i na sve njene maske bude primenjen isti skup augmentacija, sa istim parametrima. Proces augmentacije slika i maski je sledeći:

1. Učita se biblioteka
2. Definiše se augmentation pipeline: postiže se korišćenjem klase Compose. Ova klasa omogućava deinisanje niza transformacija koje se primenjuju na ulazne podatke (transformacija može biti npr. A.Flip(p=0.5) gde je p verovatnoća primene transformacije)
3. Učitavanje slike i njene maske/maski
4. Prosleđivanje slike i maske/maski pipeline-u za augmentaciju koji vraća transformisanu sliku i njenu masku/maske

## Tensorflow i Keras

*TensorFlow* [4] je open-source framework razvijen od strane Google-a koji omogućava definisanje, implementaciju, treniranje i korišćenje dubokih neuronskih mreža i ostalih modela mašinskog i dubokog učenja. Osnovni koncept TensorFlow-a uključuje korišćenje tenzora, po kojima je i dobio ime. Tenzori su višedimenzionalni nizovi koji mogu predstavljati različite tipove podataka i osnovni su elementi nad kojima se izvršavaju operacije u TensorFlow-u. Matematičke operacije nad tenzorima definišu se pomoću computational graph-ova. TensorFlow nudi više nivoa apstrakcije, tako da je moguće odabrati onaj koji najviše odgovara potrebama projekta.

*Keras* [5], s druge strane, je API visokog nivoa, koji omogućava jednostavnu izgradnju, treniranje i testiranje neuronskih mreža u TensorFlow-u. Keras pruža mogućnost korišćenja intuitivnih interfejsa i funkcija za definisanje:

- modela

- slojeva

- callback-ova

- loss funkcija

- metrika

- optimizer-a

- ...

Jedna od ključnih prednosti Kerasa je njegova jednostavnost upotrebe i fleksibilnost. Omogućava brzo definisanje prototipa modela i eksperimenata, što je izuzetno važno u istraživačkom radu i razvoju aplikacija mašinskog učenja. Takođe, Keras podržava širok spektar arhitektura neuronskih mreža, uključujući konvolucione, rekurentne mreže, itd.

TensorFlow i Keras često se koriste zajedno jer Keras olakšava izgradnju modela, dok TensorFlow pruža fleksibilnost potrebnu za njihovo treniranje i implementaciju. Kombinacija ova dva alata čini ih popularnim izborom među istraživačima u mašinskom učenju za širok spektar primena, uključujući obradu slika (computer vision), obradu prirodnog jezika, generativne modele i još mnogo toga.

Pošto se u ovom projektu za izgradnju, treniranje i testiranje modela koristi biblioteka segmentation-models, koja interno koristi Keras-ove funkcije namenjene za to, direktno korišćenjem Keras-ovog API-ja koristi se sledeće:

- callback-ovi:

  - EarlyStopping - koji zaustavlja treniranje kada loss prestane da opada (čeka 5 epoha)

  - ModelCheckpoint - koji čuva najbolji model i njegove težine

  - CSVLogger - koji čuva csv sa informacijama o treningu na kraju svake epohe

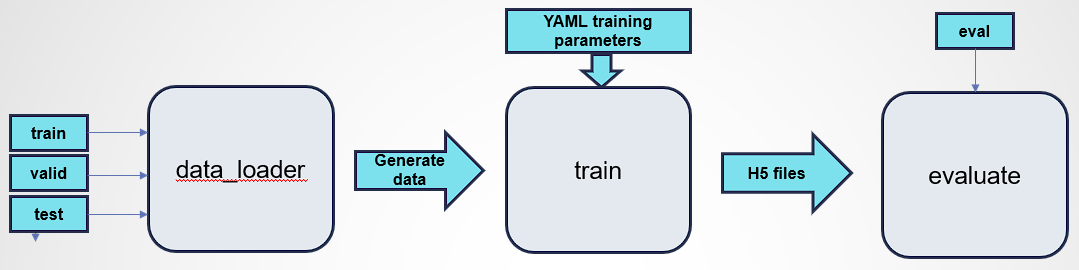
  - ReduceLROnPlateau - koji smanjuje learning rate kada loss prestane da opada (nakon 3 epohe, smanjuje ga 10 puta)

- Adam optimizer

## OpenCV

*OpenCV* [6] je moćna open-source biblioteka koja omogućava implementaciju različitih funkcija i algoritama računarskog vida u Pythonu (iako originalno pisana u C++). Sa širokim spektrom funkcionalnosti, optimizovanim performansama i podrškom za različite platforme, *OpenCV* postaje ključni alat za istraživače i industriju. Instalacija je jednostavna, a bogata dokumentacija i aktivna zajednica olakšavaju učenje i razvoj projekata. *OpenCV* pruža sve što je potrebno za implementaciju naprednih algoritama računarskog vida. U ovom projektu se uglavnom koristio za učitavanje i manipulaciju slikama.

# Implementacija



Ceo projekat je radjen u Python programskom jeziku. Korišćeni su:

* *Keras i TensorFlow* framework-ovi za trening i pripremu podataka
* *Albumentations* bilbioteka za primenu augmentacija podataka tokom treninga
* Segmentation\_models za učitavanje U-Net arhitekture, kao i adekvatnih funkcija gubitaka i metrika

Struktura projekta je sledeća:

* data:
  + train
  + valid
  + test
  + eval:
    - superpixels:
      * isolated-superpixels:
        + fire
        + nofire
      * original-full-images
* outputs:
  + training results - csv fajlovi rezultata treninga modela oblika „backbone\_loss\_metric.csv“
  + UNET weights - sačuvane težine UNET modela oblika „backbone\_loss\_metric.h5“
  + evaluation results - csv fajlovi rezulata evaluacije nad eval skupom podataka
* train\_configs – yaml fajlovi koji sadrže postavku i parametre za treniranje modela
* callbacks.py – pomoćna skripta za učitavanje callback-a iz yaml fajlova
* configuration.py – direktorijumi, podržani backbone-ovi, funkcije gubitaka i metrike
* data\_loader.py – skripta u kojoj je definisan generator podataka
* evaluate\_summarize.py – za evaluaciju modela nad eval skupom podataka
* evaluate\_all\_models.py – za evaluaciju svih modela nad eval skupom podataka
* train\_single\_model.py – sadrži funkcije za treninranje modela
* run\_single\_traning.py – za pokretanje treninga
* utils.py – pomoćne funkcije za vizuelizaciju, IoU, iscrtavanje bounding box-a

## Priprema podataka:

Podaci koji služe za trening i odabir modela za konačnu evaluaciju su u train (954 slike), valid (173 slike) i test (56 slike) direktorijumima. Svaka od slika ima svoju ground-truth masku.

Glavna skripta za učitavanje i pripremu podataka se nalazi u **data\_loader.py.** U njoj definišemo klasu *FireDataGenerator* (koja nasleđuje *keras.utils.*Sequence) za generisanje paketa slika i odgovarajućih maski kao i za primenu augmentacija na iste. Kao argumenti joj se prosleđuju sledeći parametri:

* Batch\_size: za veličinu paketa slika i maski koji se generiše
* Subset: iz kog podskupa skupa podataka se generišu slike
* Shuffle: bool indikator za mešanje podataka na kraju epohe
* Augmentations: skup transformacija koje se primenjuju na paket slika i maski za generisanje

Unutar *\_\_get\_item\_()* metode se vrši učitavanje slike i maske za generisanje, kao i njihova augmentacija (ukoliko je prosleđena) i normalizacija vrednosti piksela na opseg realnih vrednosti [0, 1]. Metod vraća paket slika i njihovih maski.

Unutar on\_epoch\_end se vrši mešanje indeksa skupa podataka na kraju epohe.

Definisana je i wrapper funkcija koja kreira instancu *FireDataGenerator* u vidu klase *get\_fire\_data\_generator,* čija je namena da osigura pravilno korišćenje augmentacija (ne dozvoljava augmentaciju podskupova podataka za validaciju i testiranje).

## Konfiguracija projekta

Unutar **configuration.py** su definisani direktorijumi za smeštanje rezultata evaluacije, kao i za skupove podataka. Defnisani su i podržane backbone arhitekture (lista string imena), metrike (dictionary koji mapira string ime u jednu od definisanih segmentation\_models metrika), loss funkcije (dictionary koji mapira string ime u jednu od definisanih segmentation\_models funkcija gubitaka), optimizatori (dictionary koji mapira string ime u jednu od definisanih keras optimizatora), augmentacije (dictionary koji mapira string ime u jednu od definisanih albumentations transformacija), i dimenzije slika (224x224). Svrha ove skripte je da mapira vrednosti parametara yaml fajlova koji sadrže konfiguraciju za treniranje i da osigura da se samo podržani parametri koriste.

## Treniranje

Funkcija za trening je unutar **train\_single\_model.py.** Njoj se prosleđuje yaml fajl koji sadrži konfiguraciju treninga modela.

Yaml fajlovi treba da budu smešetni unutar train\_configs direktorijuma i sledećeg su oblika:

* + - **backbone:** string ime backbone CNN mreže
    - **batch\_size:** veličina paketa slika i maski
    - **epochs:** broj epoha
    - **optimizer:** string ime optimizatora
    - **loss:** string ime funkcije gubitka
    - **metric:** string ime metrike
    - **callbacks:**
      * **early\_stopping:**
        + **patience:** celobrojna vrednost
        + **monitor:** string ime vrednosti koja se prati
        + **mode:** string, min ili max
      * **reduce\_lr\_on\_plateau:**
        + **patience:** celobrojna vrednost
        + **monitor:** string ime vrednosti koja se prati
        + **mode:** string, min ili max
      * **save\_model:**
        + **monitor:** string ime vrednosti koja se prati
        + **mode:** string, min ili max
        + **unet\_weights\_file:** ime .h5 fajla modela
      * **csv\_logger:**
        + **output\_csv\_file**: ime csv fajla rezultata treninga

Izabrani model je U-Net arhitektura. Ona se satoji od enkodera i dekodera, gde enkoder može biti neka od komplikovanijih CNN arhitektura za klasifikaciju bez poslednjeg FC sloja. Za implementaciju je koriscena sm.Unet klasa segmentation\_models bilioteke, kojoj je dovoljno proslediti string ime backbone arhitekture (podrazumevano se učitavaju težine dobijene pretreniranjem backbone-a na ImageNet dataset-u, što je u ovom projektu i odabrano).

Parametri se izvlače iz yaml fajla i na osnovu njih se kreira odgovarajuća U-Net arhitektura, callback-ovi i njihovi parametri, ime fajlva za čuvanje težina i rezultata treninga. Ime fajlova za čuvanje težina i rezultata treninga su oblika backbone\_loss\_metric i čuvaju se samo najbolje težine (na osnovu parametara save\_model callback-a, u ovom projektu je to izlaz funkcije gubitka validacionog skupa podataka).

Neki parametru su za potrebe ovog projekta postavljeni unapred i isti su za sve backbone-loss-metric trojke:

* + - **batch\_size:** 16
    - **epochs:** 40
    - **optimizer:** „adam“
    - **callbacks:**
      * **early\_stopping:**
        + **patience:** 5
        + **monitor:** „val\_loss“
        + **mode:** „min“
      * **reduce\_lr\_on\_plateau:**
        + **patience:** 3
        + **monitor:** „val\_loss“
        + **mode:** „min“
      * **save\_model:**
        + **monitor:** „val\_loss“
        + **mode:** „min“
        + **unet\_weights\_file:** „default“
      * **csv\_logger:**
        + **output\_csv\_file**: „default“

Trening se pokreće komandom:

python run\_single\_tranining.py –config ime\_yaml\_fajla.yaml

## Preliminarna evaluacija

Za svaki od kreiranih .h5 fajlova se vrši evaluacija nad podskupom podataka za testiranje u **preliminary\_evaluation.py.** Korišćena metrika je Intersection over Union. Za svaku sliku se uzima predikcija iz unet modela njenim propuštanjem kroz isti i poredi sa ground-truth maskom korišćenjem segmentation\_models IoUScore. Uzima se prosečna vrednost IoU nakon što se izračuna za svaku sliku. Najbolji model iz svake klase backbone arhitektura se šalje na konačnu evaluaciju.

## Konačna evaluacija

Konaćna evaluacija se vrši nad skupova podataka iz eval direktorijuma. Ovaj put se evluacija ne vrši na nivou slika kao što je to bio slučaj kod preliminarne slike, već na nivou suprepiksela (problem se svodi na klasifikaciju svih superpiksela u jednu od klasa „fire“ i „nofire“. Metrike za evaluaciju su True Positive Rate (TPR), False Positive Rate (FPR), Precision, Recall i Accuracy.

Unutar eval direktorijuma se nalaze 224x224 slike i njihovi superpikseli (smešteni u fire ili nofire direktorijum).

Unutar **evaluate\_summarized.py** se vrši evaluacija za jedan model (h5 fajl u ovom slučaju) i definisane su dve funkcije: *evaluate\_image()* i *evaluate\_on\_all\_images().*

*Evaluate\_image()* funkcija kao argumente prima putanju do slike, predikciju modela, prag tolerancije za Intersection over Union za fire superpiksele i prag tolerancije za nofire superpiksele. Unutar nje se učitava slika na osnovu njene putanje, vršit ekstrakcija iz predikcije modela na osnovu binarizacije superpiksela logičkom AND operacijom. Potom se za slučaj fire superpiksela vrši Intersection\_over\_union sa ektrakovanim superpikselom iz predikcije modela (ukoliko je veći od praga tolerancije, povećava se True Positive promenljiva, inače se povećava False Negative), a za slučaj nofire superpiksela se gleda odnos jedinica i nula preseka nofire suprepiksela i ekstrakovanog superpiksela (ukoiko je manji od praga tolerancije, povećava se True Negative a inače se povećava False Positive).

*Evaluate\_on\_all\_images()* funkcija kao argumente prima pitanju do .h5 fajla modela, prag tolerancije za Intersection over Union za fire superpiksele i prag tolerancije za nofire superpiksele. Unuatar nje se za svaku sliku vrši predikcija i poziva funkcija *evaluate\_image()*. Takođe se svaka predikcija diskretizuje prema pragu tolreancije koji može biti (0.05, 0.2, 0.4). Rezultati se sumiraju i formiraju se konačni rezultati metrika TPR, FPR, Precision, Recall i Accuracy i smeštaju u csv fajl (svakom redu podataka je pridružen i prag tolerancije disktretizacije predikcije) čije je ime formata backbone\_loss\_metric\_firethres\_nofirethres.csv. Ovi rezultati se porede sa onima iz navedenog rada u sekciji 1.

# Rezultati

Preliminarna evaluacija je dala sledeće rezultate:

1. ResNet-101:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Dice** | **BCE** | **Jaccard** | **BCE\_Dice** | **BCE\_Jaccard** |
| 0.712 | 0.63 | 0.12 | 0.708 | **0.711** |

1. MobilenetV2:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Dice** | **BCE** | **Jaccard** | **BCE\_Dice** | **BCE\_Jaccard** |
| 0.662 | 0.47 | 0.66 | 0.65 | **0.667** |

1. InceptionNetV3:

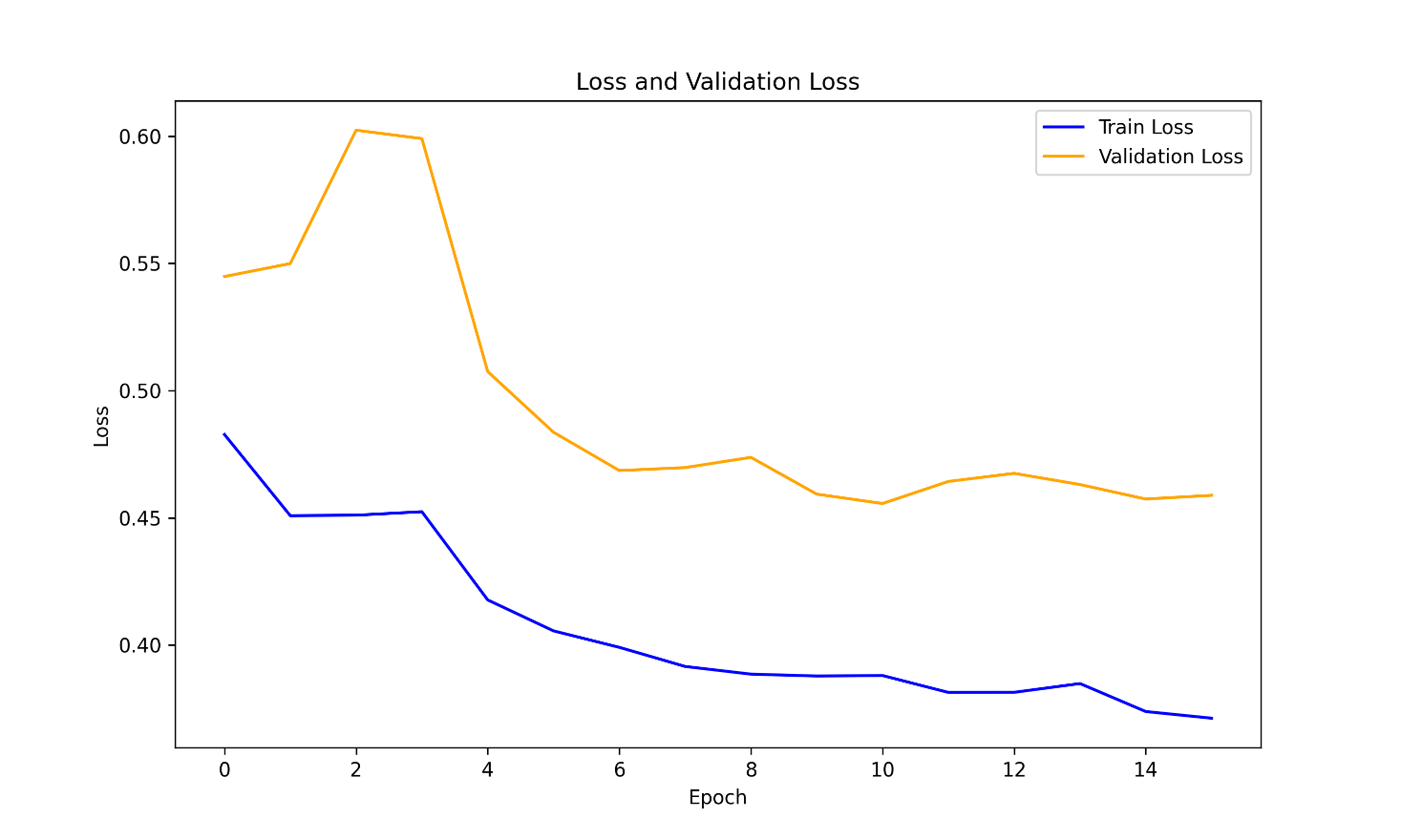
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Dice** | **BCE** | **Jaccard** | **BCE\_Dice** | **BCE\_Jaccard** |
| 0.726 | 0.63 | 0.12 | **0.7267** | 0.711 |

1. VGG-19:

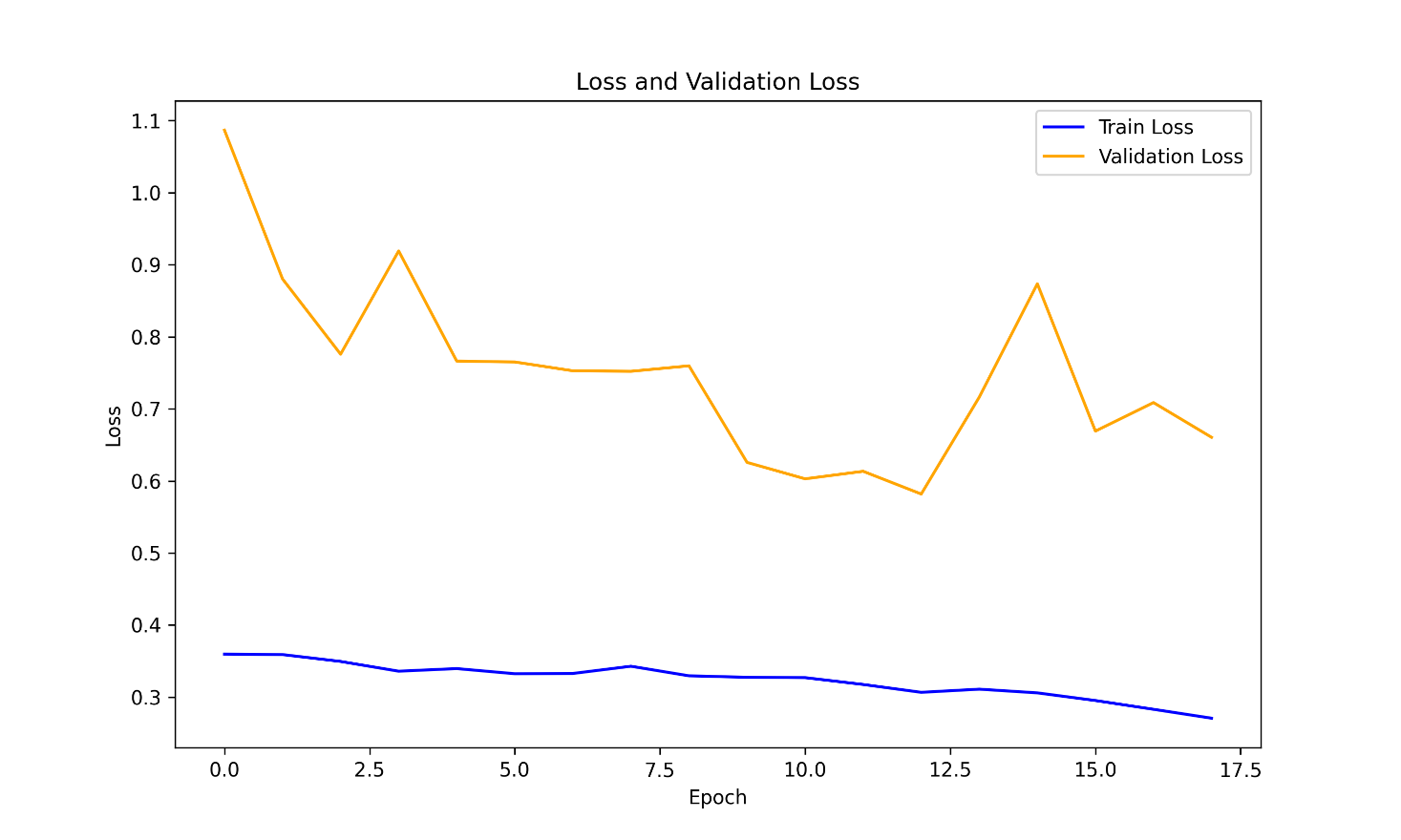
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Dice** | **BCE** | **Jaccard** | **BCE\_Dice** | **BCE\_Jaccard** |
| **0.69** | 0.601 | 0.677 | 0.14 | 0.59 |

Dati su grafici treniranja najboljih modela iz svake klase backbone-arhitektura:

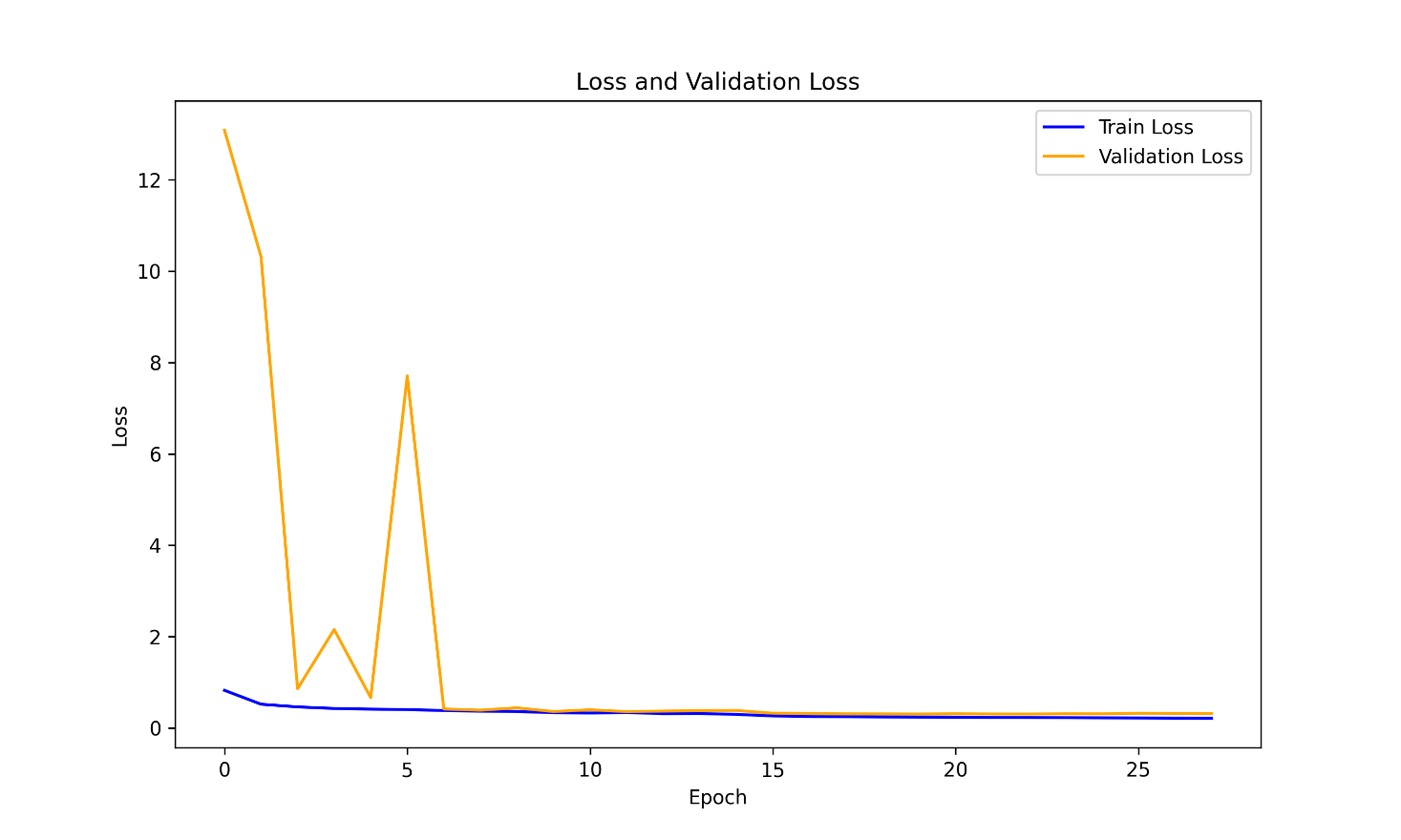
1. Resnet-101:



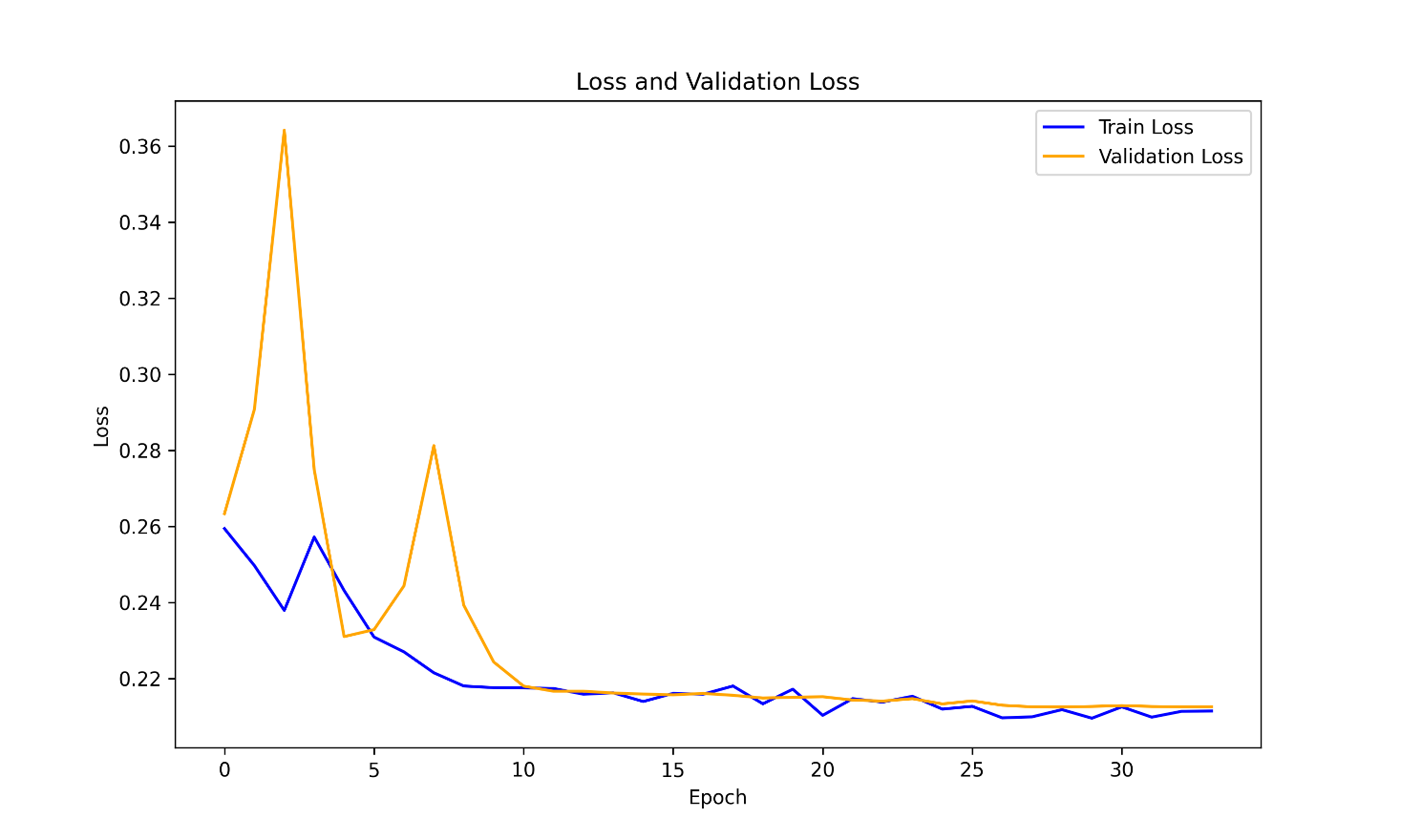
1. MobileNet-V2:



1. InceptionNet-V3:

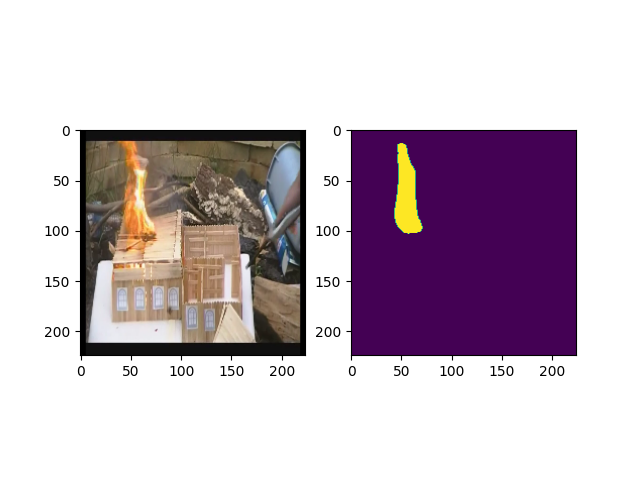


1. VGG-19:

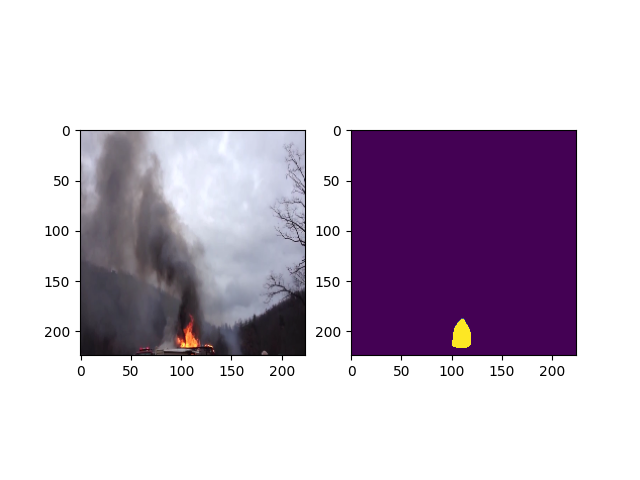


Primeri predikcija:

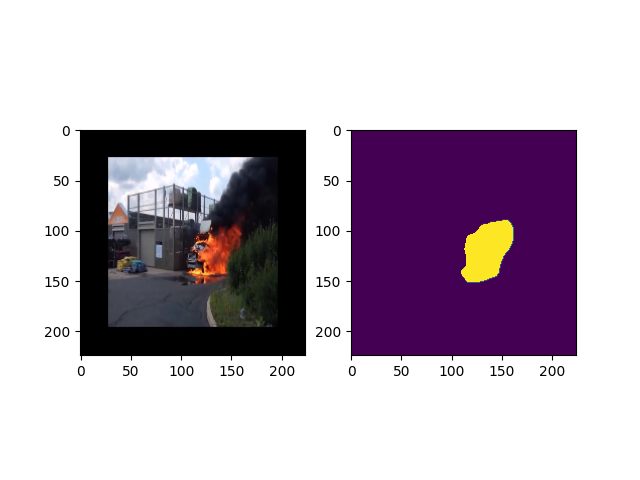
1. ResNet-101 (fire\_iou\_threshold = 0.35, nofire\_threshold = 0.5):



1. MobileNetV2: (fire\_iou\_threshold = 0.5, nofire\_threshod = 0.5):



1. InceptionNetV3 (fire\_iou\_threshold = 0.5, nofire\_threshold = 0.5):



1. VGG-19 (fire\_iou\_threshold = 0.35, nofire\_threshold = 0.5):

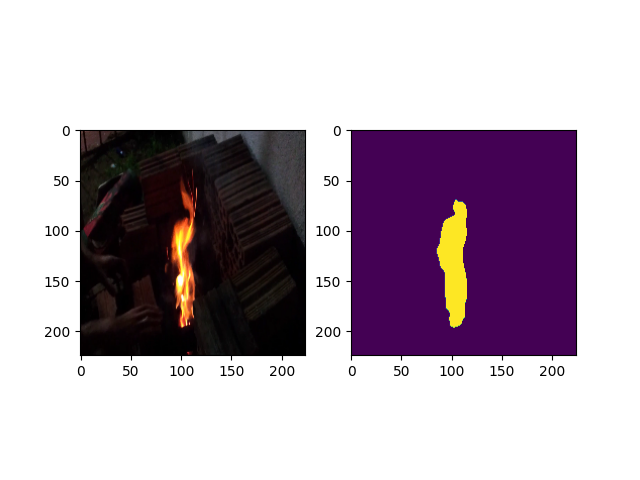


Table konačne evlauacije i poređenja sa rezultatima rada. Prag diskretizacije predikcija je svuda bio 0.4.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | TPR | FPR | Precision | Recall | Accuracy |
| Rad | **0.92** | 0.26 | **0.96** | N/A | 0.9 |
| ResNet-101 | 0.779 | **0.0179** | 0.78 | 0.779 | **0.9668** |
| MobileNetV2 |  |  |  |  |  |
| InceptionNetV3 | 0.749 | 0.0195 | 0.758 | 0.749 | 0.963 |
| VGG-19 | 0.829 | 0.0238 | 0.739 | 0.829 | 0.965 |

# Zaključak

Sve U-Net arhitekture imaju veću preciznost i FPR u odnosu na model za lokalizaciju vatre iz rada, dok isti ima bolji TPR i preciznost od svake U-Net arhitekture. Međutim, model predložen u radu za trening koristi čak 23408 slika, za validaciju 2931 slika i 21 video snimak, što daleko premašuje veličinu skupa podataka sa kojim se radilo u ovom projektu.

# Reference

[1] [iopscience.iop.org/article/10.1088/1742-6596/2278/1/012024/pdf](https://iopscience.iop.org/article/10.1088/1742-6596/2278/1/012024/pdf)

[2] [GitHub - qubvel/segmentation\_models: Segmentation models with pretrained backbones. Keras and TensorFlow Keras.](https://github.com/qubvel/segmentation_models)

[3] [Albumentations Documentation](https://albumentations.ai/docs/#getting-started-with-albumentations)

[4] [TensorFlow](https://www.tensorflow.org/)

[5] [Keras: Deep Learning for humans](https://keras.io/)

[6] [OpenCV - Open Computer Vision Library](https://opencv.org/)