Prirodno-matematički fakultet
Informatika

Projekat iz predmeta Mikroprocesorski sistemi

Detekcija šumskih požara i alarm

Profesor

Dr Aleksandar Peulić

Studenti

Vera Ranković

Ksenija Očišnik

Tamara Lukić

Sadržaj

Kratak opis projekta

Projekat *Detekcija šumskih požara i alarm* uz pomoć veštačke inteligencije vrši detekciju požara na satelitskim snimcima. Korisnik može učitati sliku, a naša aplikacija će izvršiti analizu i prikazati da li uspeva da detektuje požar, kao i procenat verovatnoće tačnosti predikcije.
Osim toga, aplikacija omogućava pokretanje alarma ukoliko je požar detektovan putem različitih metoda. Moguće metode jesu putem zvučnog alarma, slanjem mejla na uneti mejl, ili slanje binarnog signala 1 na uneti port.

Dataset satelitskih slika

Dataset koji je korišćen prilikom učenja modela je javni dataset satelitskih slika na online platformi Kaggle, koja nudi ogroman broj dataset-ova za analizu podataka i treniranje modela mašinskog učenja.

Dataset se sastoji od satelitskih slika podaljenih u 2 klase :

1. fire – 22710 slika
2. no_fire – 20140 slika

U cilju treniranja modela za detekciju šumskih požara, dataset je podeljen u 3 dela, gde se svaki od delova dalje deli na fire i no_fire :

1. Train – 70% slika – za obučavanje modela, gde model uči karakteristike slika koje prikazuju požar i one koje ne prikazuju
2. Validation – 15% slika – za proveravanje tačnosti modela na neviđenim podacima
3. Test – 15% slika – za konačnu evaluaciju modela nakon završetka treniranja za konačnu evaluaciju i merenja tačnosti modela

Model mašinskog učenja

Izabrani model za potrebe ovog projekta je ResNet50V2, što je unapređena verzija ResNet50 (Residual Networks) arhitekture za duboko učenje, posebno razvijena za efikasno treniranje veoma dubokih neuralnih mreža.

ResNet50V2 je izabran jer :

1. Ima visoke performanse na složenim dataset-ovima
2. Efikasan je za složene zadatke klasifikacija slika
3. Treniran je na ImageNet dataset-u, koji se sastoji od miliona slika iz različitih klasa
4. Sposoban je da se prilagodu novom dataset-u

Ključne biblioteke

Za potrebe ovog projekta korišćene su sledeće biblioteke

- TensorFlow/Keras

TensorFlow je jedan od najpopularnijih framework-ova za mašinsko učenje i duboko učenje, dok je Keras high-level API ove platforme. Omogućava nam da obučimo model na dataset-u i kasnije napravimo predikcije na osnovu slika.

TensorFlow/Keras je korišćem za definisanje, treniranje i evaluaciju CNN modela (ResNet50V2).

Keras API omogućava lako kreiranje i upravljanje slojevima modela, kao što su konvolucioni slojevi, slojevi za regularizaciju (BatchNormalization, Dropout), kao i dodavanje potpuno povezanih slojeva (Dense).

- Pillow (PIL)

Pillow je Python biblioteka za rad sa slikama. Korišćena je za učitavanje slika koje korisnici ubacuju u GUI aplikaciju, za manipulaciju slika (promena veličine, konvertovanje u format koji model može da koristi), kao i njihovo prikazivanje.

- NumPy

NumPy je osnovna Python biblioteka za rad sa više-dimenzionalnim nizovima i matematičkim operacijama. Korišćena je za obradu slika koje ulaze u model. Kada se slika učita, biblioteka je korišćena NumPy za konverziju slike u **niz** koji model može da obradi.

- Tkiner (tk)

Tkinter je Python biblioteka za kreiranje grafičkih korisničkih interfejsa (GUI). Korišćena je za kreiranje **GUI aplikacije** koja omogućava učitavanje slike, izvršavanje predikciju i prikazivanje rezultata. Omogućava dodavanje interaktivnih elementa, kao što su dugmići, tekstualna polja I labele, kao i prikaz ostalih elemenata aplikacije.

- Serial (pySerial)

pySerial je Python biblioteka za komunikaciju sa serijskim portovima. Korišćena je za slanje signala na definisani port.

- Pygame

Pygame je Python biblioteka za rad sa multimedijalnim aplikacijama, često korišćena za igre, zvuk i grafiku. Korišćena je za reprodukciju zvuka alarma kada se detektuje požar, čime se pruža zvučno obaveštenje.

- Smtplib

Smtplib je standardna Python biblioteka koja omogućava slanje email poruka putem SMTP protokola. Korišćena je za slanje email obaveštenja u slučaju da model detektuje požar ukoliko je opcija za slanje mejla izabrana.

- SciPy

SciPy je Python biblioteka za naučne i tehničke operacije. Korišćena je za optimizacione funkcije i dodatne matematičke operacije koje podržavaju treniranje modela.

Pokretanje projekta

1. Kloniranje projekta sa git-a

```
git clone https://github.com/ksenijaocisnik/MS_PROJEKAT.git
cd MS_PROJEKAT
```

2. Kreiranje i aktivacija virtuelnog okruženja

Virtuelno okruženje izoluje sve instalirane pakete, kako bi projekat imao sve potrebne zavisnosti bez mešanja sa sistemskim Python-om ili drugim projektima.

a. Windows
```
python-m venv venv
venv\Scripts\activate
```
b. macOS/Linux:
```
python3-m venv venv
source venv/bin/activate
```

3. Instalacija potrebnih paketa iz requirements.txt

Sve potrebne biblioteke navedene su u requirements.txt i ova komanda će ih sve instalirati, čime se osigurava da imaš pravu verziju svake biblioteke.

```
pip install-r requirements.txt
```

4. Priprema podataka (potrebna samo za treniranje i evaluaciju modela)

Potrebno je da dataset (satelitske slike) bude na odgovarajućoj lokaciji. Dataset sa Kaggle-a je jako velik, sadrži više od 40 000 slika. Upravo iz tog razloga se ne nalazi u git projektu, ali je poteban za treniranje i evaluaciju modela.

Slike je potrebno podeliti na način opisan u Dataset delu, jer je organizacija slika jako važna za pravilno treniranje i izvršavanje predikcija.

5. Pokretanje projekta

Glavna aplikacija se pokreće koristeći glavnu Python skrptu, što je potrebno uraditi komandom:

```
python main.py
```

6. Biranje opcije za alarm

Glavna skripta pokreće GUI aplikaciju

Potrebno je izabrati opciju za tip alarma ukoliko predikcija ukazuje na požar. Opcije za alarm su zvuk, slanje na uneti email i slanje na uneti port. Opcija slanja na mejl zahteva promene u kodu, gde je potrebno na adekvatno mesto uneti mejl pošiljaoca kao i lozinku, ali je bitno da se izabere email koji ne zahteva 2FA. Opcija slanja na port zahteva da postoji povezani uređaj, kako bi funkcija mogla da se izvrši.

7. Učitavanje slike

U pokrenutoj GUI aplikaciji, slika se može učitati, nakon čega se izvršava predikcija i kao i procenat sigurnosti modela u rezultat. Ukoliko je rezultat požar, pokrenuće se izabrani alarm. Podržani formati su PNG i JPEG

Struktura projekta

```
/Mikroprocesorski_Sistemi_Projekat/
|
├── /satelite_images/
|   ├── /train/                    # Sadrži slike za treniranje
|   |   ├── /fire/                 # Slike požara za treniranje
|   |   └── /no_fire/              # Slike bez požara za treniranje
|   ├── /valid/                    # Sadrži slike za validaciju
|   |   ├── /fire/                 # Slike požara za validaciju
|   |   └── /no_fire/              # Slike bez požara za validaciju
|   └── /test/                     # Sadrži slike za testiranje
|       ├── /fire/                 # Slike požara za testiranje
|       └── /no_fire/              # Slike bez požara za testiranje
|
├── /satelite_images_examples/     # Primeri slika koje možemo da učitamo u aplikaciji
|   ├── example1.jpg
|   ├── example2.jpg
|   └── example3.jpg
|
├── /alarm_sounds/                 # Zvuk za alarm u slučaju detekcije požara
|   └── strange-notification.mp3   Zvuk alarma
|
├── /venv/                         # Virtuelno okruženje za instalirane biblioteke
|
├── fire_detection.h5              # Trenirani model koji se koristi za predikciju
├── resnet50v2_weights_tf_dim_ordering_tf_kernels_notop.h5  # Težine za ResNet50V2 model
|
├── data_loader.py                 # Funkcija za učitavanje podataka (slika)
├── resnet50.py                    # Definicija CNN modela (ResNet50V2)
├── get_prediction.py              # Funkcija za izvršavanje predikcije
├── preprocess_image.py            # Funkcija za preprocesiranje slike
├── notifications.py               # Funkcije za slanje alarma (email, zvuk, signal)
├── gui.py                         # GUI aplikacija kreirana sa tkinter-om
├── train.py                       # Skripta za treniranje modela
├── evaluate.py                    # Skripta za evaluaciju modela
├── requirements.txt               # Lista potrebnih biblioteka za projekat
└── main.py                        # Glavna skripta koja pokreće GUI aplikaciju
```

Osim toga u okviru projekta, nalazi se i folder Mikrokontroler u kome se nalazi naš STM32 projekat koji služi za aktivaciju alarma.

Analiza koda i bitnih funkcija

Analiziraćemo rad našeg projekta počevši od osnovnih funkcija ka glavnoj Python skripti za pokretanje gui aplikacije.

Python aplikacija

1. Učitavanje slika za proces učenja : data_loader.py, load_data funckija

Ova funkcija nam omogućava da učitamo podatke iz foldera satelite_images. Koristi ImageDataGenerator iz biblioteke keras kako bismo slike iz dataset-a pripremili za proces učenja. Vraća tuple, odnosno generator slika za generisanje slika za trening, validaciju i testiranje.

Kako bismo kreirali naš generator slika, potrebno je da normalizujemo sliku, tj. da piksele slike koji se kreću od 0 do 255 pretvorimo u vrednosti od 0 do 1. Osim toga potrebno je da defnišemo izmenu slika, kako bismo osigurali da naš model može lako da generalizuje unete slika. Stoga postavljamo paramtere za nasumično rotiranje slike od 20 stepeni, kao i nasumično horizontalno okretanje.

```
image_data_generator = ImageDataGenerator(rescale=1./255, rotation_range=20, horizontal_flip=True)
```

Sledeći korak jeste kreiranje generatora za svaki deo procesa (train, valid, test), zbog čega koristimo naš kreiran generator i njegovu funkciju flow_from_firectory, koja nam omogućava da slike učitavamo direktno iz direktorijuma postepeno u grupama(batch).

Primer za train_generator:

```
train_generator = image_data_generator.flow_from_directory(
    train_dir,
    target_size=(IMAGE_HEIGHT, IMAGE_WIDTH),
    batch_size=BATCH_SIZE,
    class_mode='binary'
)
```

Bitno je napomenuti da je naša klasifikacija binarna, jer slike definišemo kao fire/no_fire, pa je class_mode koji prosleđujemo našem generator *binary*.

Isti process ponavljamo i za generator validacionih i testih slika.

2. Kreiranje modela : resnet50.py, create_model funkcija

Ova funkcija nam služi da kreiramo CNN (Convolutional Neural Network) model na osnovu ResNet50V2 arhitekture. Sve potrebne funkcije se nalaze u keras biblioteci.

U funckiji create_model prvo je potrebno da učitamo pre-trained ResNetV2 model. Kako bismo ga učitali potrebno je da mu prosledimo već definisane težine, koje se nalaze u resnet50v2_weights_tf_dim_ordering_tf_kernels_notop.h5.

Takođe, potrebno je da definišemo include_top paramtar kao false, jer ne želimo da uključimo poslednje slojeve već naučenog modela, već želimo da dodamo nove kako bi naš model bio sposoban da se prilagodu našem zadataku.

Input_shape definišemo kao tri vrednosti, a tu su visina, širina i broj boja (RGB) slike.

```
base_model = ResNet50V2(weights='resnet50v2_weights_tf_dim_ordering_tf_kernels_notop.h5', include_top=False, input_shape=input_shape)
base_model.trainable = False
```

Još jedna jako bitna svar je zamrzavanje osnovnih slojeva originalnog modela, kako bi bismo samo mogli da se nadogradimo na to znanje dodavajući samo gornje slojeve.

Kada smo učitali i podesili osnovni model, sledeći korak jeste dodavanje prilagođenih slojeva. Potrebno je da smanjimo dimenziju podataka koristeći pool-ovanje, tj. tako što uzimamo prosečnu vrednost svih piskela u svakoj od 224*224(visina*širina) karakteristika koje je model već naučio. Ovo smanjuje našu kompleksnost podataka pre nego što ih prosledimo gustim slojevima. Ovaj proces postižemo pomoću funkcije GloablAveragePooling2D().

Zatim, dodajemo izlazni sloj za binarnu klasifikaciju. Korsitimo funkciju Dense, gde je prvi paramter broj neurona, što je kod nas 1 i aktivacionu funkciju sigmoid jer koristimo binarnu klasifikaciju. Sigumoid aktivaciona funkcija vraća verovatnoću između 0 i 1. Funkciji dense prosleđujemo i naš pooled_output.

```
base_model_output = base_model.output
pooled_output = GlobalAveragePooling2D()(base_model_output)
# Koristimo 1 neuron, jer je klasifikacija binarna i verovatnocu pozara predstavljamo sigmoid funkcijom koja
#   verovatnocu preslikava u broj izmedju 0 i 1 :
#       ako je blize 1 -> NO_FIRE
#       ako je blize 0 -> FIRE
output = Dense(1, activation='sigmoid')(pooled_output)
```

Sledeći korak jeste kreiranje i kompajliranje modela.

Pri kreiranju modela potrebno je da izvršimo spajanje njegovih već naučenih slojeva, sa našim dodatim slojevima. Kada je model kreiran, vršimo i kompajliranje.

Funkciji compile prosleđujemo adam optimizator, koji automatski pilagođava brzinu učenja tokom treniranja, zatim loss (f-ja greške) koju definišemo kao binary_vrossentroppy koja se koristi u binarnim klasifikacionim procesima jer meri razliku između stvarnih, tj. pravilnih vrednosti i onih koje model predviđa. Osim toga, definišemo i metriku kvaliteta modela, što je kod nas tačnost predikcije.

```
# Potrebno je da spojimo ono sto model vec zna sa slojevima koje smo mi dodali
model = Model(inputs=base_model.input, outputs=output)
# Kompajliramo model, gde kvalitet modela merimo po tacnosti
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

return model
```

3. Treniranje modela : train.py, train_model funckija

Ovaj fajl sadrži funkciju za treniranje modela. Ova funkcija sadrži ceo proces treniranja modela, od učitavanja podataka do čuvanja modela nakon treniranja.

U funkciji train_model pozivamo prethodno definisanu funkciju create_model za kreianje modela i load_data za dobijanje generatora slika za trening i validaciju.

```
# Kreiramo model
model = create_model()
# Potreni su nam generatori
train_generator, val_generator, test_generator = load_data()
```

Pre nego što pozovemo funkciju model, potrebno je da definišemo callback za rano zaustavljanje ako ima potrebe.

- Early stopping – prati gubitak na validacionom skupu (val_loss) i zaustavlja treniranje modela ako se on ne poboljša u periodu od 3 epohe. Takođe, u stanju je da vrati model na najbolju verziju sa najmanjim gubitkom.

Zatim treniramo model koristeći funkciju model.fit(). Prosleđujemo joj generator slika iz train skupa, definišemo broj epoha(10 u našem slučaju), generator slika iz valid skupa i niz callback-ova.

Nakon što je model istreniran sledi čuvanje u napred definisanu putanju.

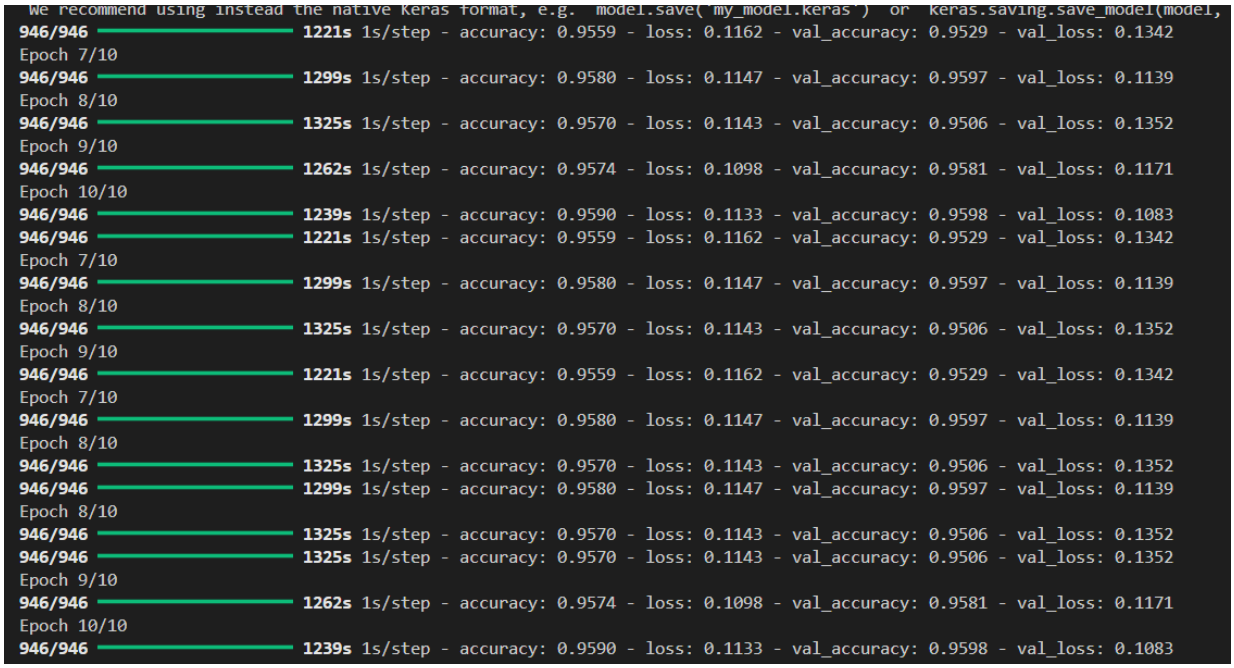
```
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
model.fit(train_generator, epochs=EPOCHS, validation_data=val_generator, callbacks=[early_stopping])

# Cuvamo istrenirani model
model.save(model_save_path)
print(f"\033[92mModel je sačuvan: {model_save_path}\033[0m")
```

U fajlu train.py se nalazi main funkcija koja pokreće process treniranja modela pokretanjem komande

```
python train.py
```

Vrednosti kroz iteracije treniranja:



4. Evaluacija modela : evaluate.py

Ovaj fajl sadrži samo main funkciju koja služi za evaluaciju istreniranog modela. Izračunava gubitak i tačnost modela koristeći podateke iz test skupa slika i loguje ih u konzolu.

Pokreće se komandom :

```
python evaluate.py
```

Kako bismo izvršili proces evaluacije, potrebno je da prethodno istrenirani i sačuvani model učitamo i pokrenenmo funkciju model.evaluate() kojoj prosleđujemo generator slika iz testnog skupa koji dobijamo iz prethodno definisane load_data funkcije.

Model.evaluate() vraća 2 vrednosti nad vrednostima iz testnog skupa:

1. Loss (greška/gubitak) – od 0 do 1 – što je gubitak manji, preciznost je bolja.
2. Accuracy (tačnost) – od 0 do 1 - pokazuje procenat ispravno klasifikovanih slika

```python
if __name__ == '__main__':
    # Potrebno je da ucitam kreirani model
    fire_detection_model = load_model('fire_detection.h5')
    # Potreban je generator slika za test folder
    _, _, test_generator = load_data()

    # Evalucaija modela:
    test_loss, test_accuracy = fire_detection_model.evaluate(test_generator)

    # Loss -> gubitak
    # Pokazuje nam koliko nas model gresi na datom skupu podataka
    print(f"\033[94mTest loss: {test_loss:.4f}\033[0m")
    # Accuracy -> Tacnost
    # Pokazuje nam procenat ispravnih predikcija
    print(f"\033[94mTest accuracy: {test_accuracy:.4f}\033[0m")
```

Dobijene vrednosti za naš model su:

1. Loss = 0.0952 – možemo da smatramo da naš model retko greši
2. Accuracy = 0.9651 – možemo da smatramo da naš model dovoljno dobro generalizuje slike i tačno predviđa klasifikaciju novih podataka/satelitskih slika.

5. Prilagođavanje novih slika za predikciju : preprocess_image.py

Istoimena funkcija u ovom fajlu služi za prilagođavanje slike za koju je potrebno izvršiti predikciju, jer naš model prihavata podatke samo u odgovarajućem formatu.

Funkciji je potrebno da prosledimo putanju na kojoj se nalazi. Zatim učitavamo sliku, pomoću funkcije load_img kojoj prosledjujemo datu putanju i veličinu koja nam je potrebna.

Zatim, pretvaramo sliku u niz piksela pomoću funkcije img_to_array i normalizujemo na vrednosti od 0 do 1, tako što dobijeni niz delimo sa brojem piksela, odnosno 255.

Kada smo dobili niz u odgovarajućem formatu, potrebno je da dodamo i batch dimenziju, obzirom da je model naučen na grupama slika. Ovo radimo pomoću funckije iz numpy biblioteke, expand_dims kojoj prosleđujemo naš niz i axis=0 za dodavanje batch dimenzije na početak niza.

Niz koji vraćamo je u formatu (1, 244, 244, 3) – grupa od jedne slike, sa visinom i sirinom od 244 piskela i 3 kanala za broje (RGB).

```
# Funkcija za obradu slike kakvu nas model moze da prepozna
def preprocess_image(image_path):
    # Manjmo sirinu i visinu
    img = load_img(image_path, target_size=(IMAGE_HEIGHT, IMAGE_WIDTH))

    # Normalizujemo piksele na opseg izmedju 0 i 1
    img_array = img_to_array(img) / 255.0

    # Potrebno je da dodamo batch dimenziju, sto je u nasem slusaju 1 (grupa koja se sastoji od jedne slike)
    # axis = 0 -> batch dimenzija se dodaje na pocetak
    # nas niz izgleda ovako : (1, 224, 224, 3) : grupa od jedne slike, visina, sirina, broj boja
    img_array = np.expand_dims(img_array, axis=0)

    return img_array
```

6. Notifikacije/Alarm : notifications.py

U ovom fajlu se nalaze tri 3 funkcije za pokretanje alrama ukoliko je naš model odredio da se na slici nalazi požar.

1. Funkcija sa slanje unetog mejla : send_email_alert

Koristimo biblioteku smtplib za pokretanje servera koji ce nam omoguciti da posaljemo mejl. Takođe potrebno je da se unutar koda definiše email pošiljaoca kao i lozinka. Ovo će nam omogućiti server da se prijavi na mejl i pošalje.

U msg varijabli definišemo poruku koju želimo da pošaljemo. Zatim, ulazimo u try/catch blok u kome je prvi korak da kreiramo SMTP objekat za komunikaciju sa serverom i aktiviramo enkripciju komunikacije sa serverom pomoću TLS (Transport Layer Security).

Drugi korak jeste logovanje pomoću funkcije server.login kojoj prosleđujemo email pošiljaoca i lozinku. Bitno je i napomenuti da email pošiljaoca koji je definisan nema omogućeno 2FA zaštitu kako bi logovanje bilo uspešno.

Treći korak jeste slanje emaila pomoću funkcije sever.sendemail kojoj prosleđujemo adresu pošiljaoca, email na koji šaljemo našu poruku i poruku koju smo definisali u msg varijabli.

Konačno, nakon slanja mejla, zatvaramo konekciju sa serverom.

```python
# Funkcija za slanje email-a
def send_email_alert(email):
    # Uneti mejl (potreban je mejl koji nema omogucenu 2FA)
    sender_email = "..."
    # Unesti lozinku
    password = "..."

    msg = f"Subject: ALARM: Požar Detektovan!\n\nPožar je detektovan na satelitskom snimku."
    try:
        # Kreiramo SMTP objekat za komunikaciju sa serverom
        server = smtplib.SMTP('smtp.gmail.com', 587)
        # Zapocinjemo Transport Layer Security sesije za sifrovanje
        server.starttls()
        # Logujemo se
        server.login(sender_email, password)
        # Saljemo email na unetu adresu sa porukom koju smo definisali kao msg
        server.sendmail(sender_email, email, msg)
        # Zatvaramo konekciju sa serverom
        server.quit()
        print(f"Email je poslat na adresu: {email}!")
    except Exception as e:
        print(f"Error: slanje email-a: {e}")

# Funkcija za reprodukciju zvuka
```

2. Funkcija za aktivacija zvučnog signala : play_alarm_sound() funkcija

Za aktivaciju zvučnog signala koristimo biblioteku pygame.

Potrebno da pokrenemo player funckijom pygame.mixer.init(), zatim da učitamo alarm pomoću funkcije pygame.mixer.music.load() kojoj prosleđijemo putanju do željenog zvuka i da na kraju pomoću funkcije pygame.mixer.music.play() pustimo zvuk.

```python
# Funkcija za reprodukciju zvuka
def play_alarm_sound():
    pygame.mixer.init()
    pygame.mixer.music.load('./alarm_sounds/strange-notification.mp3')
    pygame.mixer.music.play()
```

3. Funkcija za slanje signala uređaju preko serijskog porta : send_signal_to_device funkcija

Za potrebe ove funkcije koristimo biblioteku Serial.

Naša funkcijia prima naziv porta u string formatu, zatim ulazi u try/catch blok gde pokušava da otvori serijski port sa zadatom brzinom od standardnih 9600bps koristeći funkciju serial.Serial(port, 9600).

Sledeći korak jeste slanje binarnog podataka 1 koristeći funkciju write(b'1').

Naš mikrokontroler je podešen tako da čeka naš signal, kako bi započeo blinkanje diode. Ukoliko uspešno primi naš signal, on će nam vratiti i poruku. Upravo zbog toga, potrebno je da sačekamo odgovor. To možemo uraditi pomoću funkcije ser.readline, ali osim toga moramo da izvršimo i dekodiranje poruke koju zatim ispisujemo u konzolu.

Završavamo sa našom funkcijom tako što zatvaramo port, funckijom ser.close().

```python
# Funkcija za slanje signala na uređaj preko serijskog porta i primanje povratne poruke
def send_signal_to_device(port):
    try:
        # Port i brzina prenosa podataka, standardno 9600
        # Dodajemo timeout za čitanje
        ser = serial.Serial(port, 9600, timeout=1)
        # Saljemo broj 1 kao signal uređaju, u binarnom formatu
        ser.write(b'1')
        print(f"Poslat signal na {port}.")
        # Čitanje povratne poruke sa uređaja i dekodiranje u string
        response = ser.readline().decode('utf-8').strip()
        print(f"Povratna poruka sa uređaja: {response}")

        # Zatvaramo serijski port
        ser.close()
        print(f"Povratna poruka sa uređaja: {response}")
    except serial.SerialException as e:
        print(f"Error : otvaranje porta: {e}")
    except Exception as e:
        print(f"Error: {e}")
```

Osim funkcija za aktivaciju izabranog alarma, u ovom fajlu se nalazi i funkcija koja kontroliše koji će alarm biti aktiviran, tako što proverava vrednost poslatog parametra za tip alarma koji dolazi sa gui aplikacije. Pored toga, prosleđujemo joj i vrednost polja za unos email-a i porta.

```python
# Funkcija koja se poziva na osnovu izabira radio button-a
def trigger_alarm(alarm_type, email, port):
    if alarm_type == "email":
        if email:
            send_email_alert(email)
        else:
            print("Nije uneta email adresa.")
    elif alarm_type == "sound":
        play_alarm_sound()
    elif alarm_type == "device":
        if port:
            send_signal_to_device(port)
        else:
            print("Nije unet port za slanje signala.")
```

7. Određivanje predikcije : get_prediction.py i istoimena funkcija

Jako bitan korak u računanju predikcije jeste i funkcija koja nam služi da dobijanje rezultata da li je požar detektovan ili ne.

Funkcija prima putanju do slike koje je učitana. Zatim učitava naš istreniran model pomoću funkcije load_model, zatim određuje img_array koristeći funkciju preprecess_image.

Sledeći korak jeste pravljenje predikcije. To vršimo pomoću funkcije model.predict.

Nakon dobijanja rezutata, poredimo broj, kako bismo odredili da li je manji, jedan ili veći od 0.5 (granica odluke, jer imamo binarnu klasifikaciju). Pored rezultata, određujemo i stepen sigurnosti u rezultat i vraćamo tuple : predikcija, sigurnost u predikciju.

```
# Vracamo rezultat
# Ovakva raspodela vrednosti binarne klasifikacije je ishod redosleda trening skupova (prvo fire, zatim no_fire)
if prediction < 0.5:
    result = 'fire'
    probability = (1 - prediction) * 100
else:
    result = 'no_fire'
    probability = prediction * 100

return result, probability
```

8. Kreiranje gui aplikcije : gui.py

Glavne biblioteke koje koristimo u ovom fajlu su tkiner za kreiranje GUI aplikacije i PIL koja nam koristi za učitavanje slike.

Postoje 2 glavne funkcije u ovom fajlu i 2 pomoćne.

1. Funkcija za kreiranje gui aplikacije : create_gui

Prvi korak jeste kreiranje glavnog prozora aplikacije i postavljanje naslova. Kreiranje vršimo funkcijom tk.Tk().

```
# Kreiramo glavni prozor aplikacije i dodeljujemo naziv
root = tk.Tk()
root.title("Detekcija šumskog požara")
```

Zatim konfigurišemo aplikaciju, tako što postavljamo boju, centriramo je u prozoru u kojem se otvara, postavljamo pozadinu, umećemo naslov kao komponentu i definišemo layout aplikacije koristeći pomožnu funkciju center_window i funkcije koje se nalaze u tk biblioteci.

```
# Definišemo dimenzije naše aplikacije i centriramo aplikaciju na sredinu ekrana
center_window(root, 600, 800)

# Postavljamo svetlo sivu pozadinu
root.configure(bg="#f0f0f0")

# Definišemo prostor kolone i reda
root.grid_columnconfigure(0, weight=1)
root.grid_rowconfigure(1, weight=1)

# Naslov aplikacije koji se dodaje u glavni prozor (root)
# Postavljamo ga u nultu kolonu i nulti red naše mreže (grid)
title_label = tk.Label(root, text="Detekcija šumskog požara", font=("Arial", 20), bg="#f0f0f0")
title_label.grid(row=0, column=0, padx=10, pady=20)
```

Sledeći korak jeste definisanje mesta gde ćemo prikazati učitanu sliku i predikciju. Kreirane Label-e smeštamo u promenjivu panel i title_label tako da nakon učitavanja slike možemo da je prikažemo i nakon određivanj predikcije umetnemo odgovor.

```
# Definišemo mesto za sliku i njene dimenzije
panel = tk.Label(root, bg="white", width=300, height=300)
panel.grid(row=1, column=0, padx=10, pady=20)

# Definišemo mesto za prikazivanje rezultata
result_label = tk.Label(root, text="Rezultat će biti prikazan ovde", font=("Arial", 16), bg="#f0f0f0")
result_label.grid(row=3, column=0, padx=10, pady=30)
```

Potrebno je i da definišemo radio dugmiće za odabir alarma, gde je default vrednost zvuk. Tokom kreiranja, potrebno je i da prosledimo komandu za kontrolisanje promene select opcije. To vršimo pomoću funkcije toggle_fields kojoj je potrebno da prosledimo tip alarma, tekst label za unos email polja, input label za email, tekst label za unos porta, input label za port.

Pomoćna funkcija toggle_fields kontroliše koje input polje će prikazati na osnovu odabira alarma.

```
# Opcije za odabir alarma (radio buttons), gde ćemo definisati default vrednost kao sound
# jer je za email potrebno definisati email posiljaoca i sifru, a za device je potebno da je neki uredjaj povezan za port koji zelimo
alarm_type = tk.StringVar(value="sound")
```

```
# Label za odabir tipa alarma
tk.Label(root, text="Izaberite tip alarma:", font=("Arial", 14), bg="#f0f0f0").grid(row=4, column=0, padx=10, pady=10, sticky="w")

# Kreiramo labelu i polje za email
email_label = tk.Label(root, text="Unesite email adresu:", font=("Arial", 12), bg="#f0f0f0")
email_entry = tk.Entry(root, font=("Helvetica", 12), width=30)

# Kreiramo labelu i polje za port
port_label = tk.Label(root, text="Unesite serijski port (npr. COM3):", font=("Arial", 12), bg="#f0f0f0")
port_entry = tk.Entry(root, font=("Helvetica", 12), width=30)
```

```
# Radio dugmići za tip alarma
tk.Radiobutton(root, text="Reprodukuj Zvuk", variable=alarm_type, value="sound", bg="#f0f0f0", font=("Arial", 12), command=lambda: toggle_fie
tk.Radiobutton(root, text="Pošalji Email", variable=alarm_type, value="email", bg="#f0f0f0", font=("Arial", 12), command=lambda: toggle_field
tk.Radiobutton(root, text="Pošalji Signal na Uređaj", variable=alarm_type, value="device", bg="#f0f0f0", font=("Arial", 12), command=lambda:
```

Potrebno je i definisati dugme za učitavanje željene slike. Komanda koja se prosleđuje je funkcija upload_image kojoj prosleđujemo mesto gde učitanu sliku treba da ubacimo, mesto gde treba da ažuriramo rezultat i procenat sigurnosti u predikciju, obabranu opciju za alarm, uneti email i port.

```
# Definišemo dugme za upload slike
upload_btn = tk.Button(root, text="Učitaj sliku", font=("Arial", 14), bg="#113163", fg="white", width=20, command=lambda: upload_image
upload_btn.grid(row=2, column=0, padx=10, pady=10)
```

Gui aplikciju na kraju pokrećemo pomoću funkcije mainloop().

```
# Pokrećemo glavnu petlju tkinter gui aplikacije
root.mainloop()
```

2. Funkcija za učitavanje slike : upload_image

Funkcjia koju pozivamo klikom na dugme *Učitaj sliku* , otvaramo dialog pomoću funkcije filedialog.askopenfilename() koja nam vraća putanju do učitane slike. Sliku je zatim potrebno otvoriti i promeniti joj veličinu i broj piksela. Osim toga, moramo da je pretvorimo i u format koji tkiner biblioteka može da podrži.

```
# Funkcija za upload slike i prikaz rezultata
def upload_image(panel, result_label, alarm_type, email, port):
    # Otvaramo prozor za biranje slike i ucitavamo izabranu sliku preko path-a
    file_path = filedialog.askopenfilename()
    img = Image.open(file_path)

    # Menjamo velicinu slike kako bismo je prikazali u velicinu u kojoj ce biti i obradjena
    # To ukkljucuje i promenu broj piksela, pa moramo da vrsimo Resmpling
    img = img.resize((IMAGE_HEIGHT, IMAGE_WIDTH), Image.Resampling.LANCZOS)
    # Vrsinu promenu formata u onaj koji biblioteka moze da podrzi
    img = ImageTk.PhotoImage(img)
    # Ubacujemo sliku u gui, tacnije u panel de0
    panel.config(image=img)
    # Zadrzavamo referencu na sliku kako bimo mogli i dalje da je prikazujemo (kako je ne bi pokupio garbage collector)
    panel.image = img
```

Nakon učitavanja slike, potrebno je da je pošaljemo funkciji get_prediction koja će nam vratiti rezultat i sugurnost u rezultat.

Ukoliko je predikcija da postoji požar, moramo da aktiviramo alarm pomoću funkcije trigger_alarm i promenimo prikaz rezultata u gui aplikaciji.

```
# Ažuriramo result_label-a tako sto update-ujemo gui sa potrebnim tekstom i pokrecemo alarm
# Alarm se pokrece tako sto pozivamo funkciju trigger_alarm koja ce na osnovu vrednosti alarm_type odrediti koja funkcija se poziva
print(f"\033[94mresult: {result}\033[0m")
if result == 'fire':
    trigger_alarm(alarm_type, email, port)
    result_label.config(text=f"OPASNOST! Požar detektovan!\nVerovatnoća: {probability:.2f}%", fg="red")
else:
    result_label.config(text=f"BEZBEDNO: Nema požara.\nVerovatnoća: {probability:.2f}%", fg="green")
```

9. Pokretanje glavne aplikacije : main.py

U main funkciji ovog fajla nalazi se kod za pokretanje aplikacije. Pozivamo funkciju create_gui koja nam pokreće GUI aplikaciju i vodi računa o unetim poljima, učitavanju slike i pozivanje funkcije za određivanje rezultata.

```
from gui import create_gui

# Glavna funkcija koja treba da pokrene mini gui aplikaciju

if __name__ == "__main__":
    create_gui()
```

STM32 projekat

Za potrebe našeg projekta izabrali smo STM32F103C8Tx zbog jednostavne UART konfiguracije. Ovaj kontroler će komunicirati sa našom Python aplikacijom putem serijskog porta (UART) i njegov zadatak biče kontrolisanje LED diode na osnovu primljenih signala.

1. Konfiguracija

- Koristimo UART jer je jednostavan za implementaciju I omogućava lako slanje i primanje podataka između računara i mikrokontrolera
- Pinovi za komunikaciju su PA9(TX) za slanje podataka i PA10(RX) za primanje podataka
- Baud rate postavljamo na 9600, što je standardna brzina komunikaciju koju smo već i definisali u našoj Python aplikaciji
- GPIO PA5 korismo za slanje signala LED diode

2. Generisanje i prilagođavanje koda

- Funkcija blink_LED

Ova funkcija kontroliše blinkanje LED diode na PA5 pinu. Blinkanje traje 10 puta, sa pauzom od 500 ms između svake promene stanja. Koristimo HAL_GPIO_TogglePin kako bismo menjali stanje (uključeno/isključeno) na PA5 pinu. Nakon završenog blinkanja, LED se isključuje tako što se PA5 postavlja na niski nivo pomoću funkcije HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET).

```c
void blink_LED(void) {
    for (int i = 0; i < 10; i++) {  // LED će blinkati 10 puta
        HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);  // Uključi/isključi LED
        HAL_Delay(500);  // 500ms delay između promena
    }
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);  // Isključi LED nakon blinkanja
}
```

- Funkcija send_alarm_message

Ova funkcija nam služi za šalje tekstualne poruku preko UART-a. HAL_UART_Transmit() šalje poruku preko UART-a. Koristi se UART koji je prosleđen kao parametar (huart).
U promenljivoj msg definišemo poruku koju vraćamo kao odgovor na primljeni signal 1.

Takođe name je potrebna i veličina poruke kao paramtetar, koju dobijamo kao strlen(msg) iz biblioteke string.h.

```c
void send_alarm_message(UART_HandleTypeDef *huart) {
    char msg[] = "ALARM: LED dioda je aktivirana.\r\n";
    HAL_UART_Transmit(huart, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);  // Pošalji poruku preko UART-a
}
```

- Glavna while petlja

Petlja radi beskonačno i čeka da primi jedan bajt podataka preko UART-a, a za to nam služi funkcija HAL_UART_Receive.

Ukoliko primi signal 1, aktivaramo funkciju za blinkanje diode (blink_LED), kao i za slanje povratne poruke (send_alarm_message).

Ako primimo bilo koji drugi signal, diode se isključuje.

```c
/* USER CODE BEGIN WHILE */
while (1)
{
    uint8_t receivedData[1];  // Čuva primljene podatke preko UART-a
    if(HAL_UART_Receive(&huart1, receivedData, 1, HAL_MAX_DELAY) == HAL_OK)
    {
        if(receivedData[0] == '1')
        {
            blink_LED();  // Blinkanje LED diode
            send_alarm_message(&huart1);  // Slanje poruke preko UART-a
        }
        else
        {
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);  // Isključi LED ako nije poslat signal '1'
        }
    }
}
/* USER CODE END 3 */
```

GUI aplikacija

Figure 1: Počeno stanje

Figure 2 : Prediction : fire, sound alarm

Figure 1 Predicrion : no fire, signal alarm

Figure 2 Prediction : no fire, alarm email