FAKULTA STROJNÍHO INŽENÝRSTVÍ
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

RAI - Detekce zdravosti listů rostlin pomocí CNN

2024/2025

Roman Marek (xmarek74)

Obsah

1 Úvod

Detekce nemocí rostlin je klíčovým úkolem v zemědělství, který má významný dopad na výnosy plodin a udržitelnost zemědělské produkce. Včasné a přesné rozpoznání zdravotního stavu rostlin umožňuje rychlou reakci na infekce a minimalizuje ztráty způsobené chorobami. Tradiční metody diagnostiky jsou často časově náročné, vyžadují odborníky a nejsou škálovatelné.

S nástupem umělé inteligence, především hlubokého učení a konvolučních neuronových sítí (CNN), se otevřely nové možnosti automatické detekce a klasifikace nemocí rostlin přímo z obrazových dat. Tyto metody umožňují rychlou, přesnou a automatizovanou analýzu, která může být implementována i v mobilních nebo desktopových aplikacích.

Cílem tohoto projektu je vytvořit desktopovou aplikaci pro detekci zdravosti rostlin na základě analýzy snímků listů pomocí konvolučního neuronového modelu. Aplikace umožňuje uživateli načíst obrázek rostliny a model a následně provést klasifikaci, zda je rostlina zdravá nebo nemocná. Krom samotné aplikace však byly implementovány i dílčí skripty jako `parseDataset.py` a `main.py`, které se dají využít i bez samotné aplikace.

2 Použité technologie

2.1 Konvoluční neuronové sítě (CNN)

Konvoluční neuronové sítě jsou dnes jedním z nejvýznamnějších nástrojů pro analýzu a klasifikaci obrazových dat. Jsou navrženy tak, aby automaticky extrahovaly důležité rysy z obrázků bez potřeby ručního navrhování těchto příznaků. Využívají speciální typ vrstev – konvoluční vrstvy, které provádějí operaci skládání mezi vstupním obrazem a sadou filtrů, jejichž váhy se učí během trénování. Výhoda CNN spočívá v jejich schopnosti zachytit prostorové vztahy v obraze – nejprve identifikují jednoduché prvky (například hrany), a postupně na vyšších vrstvách složitější struktury (například tvary, části objektů nebo celé objekty). Architektura CNN se obvykle skládá z těchto bloků:

- Konvoluční vrstvy - extrahují rysy z obrazu
- Pooling vrstvy - redukují rozměry a zvyšují robustnost
- Aktivační funkce - umožňují modelu učit se složitější vzory
- Plně připojené vrstvy - slouží ke klasifikaci rysů
- Dropout či Batch Normalization - zabraňují přeučení

2.2 Keras a TensorFlow

Pro vývoj modelu byla využita knihovna `Keras`, která běží nad frameworkem `TensorFlow`. Ten umožňuje efektivní běh na různých platformách - od CPU až po mobilní zařízení. Je určen jak pro výzkumné účely, tak i pro nasazení do produkce. Jeho mírnou nevýhodou je však složitější použití. Z tohoto důvodu jsem se uchýlil právě ke knihovně `Keras`, která nabízí vyšší abstrakci pro budování a trénování neuronových sítí. Pro tvorbu modelů využívá objektově orientovaný přístup a minimalizuje množství nutného kódu.

2.3 PyQt5

PyQt5 je populární knihovna pro tvorbu grafických rozhraní v programovacím jazyce Python. Je napsána v jazyce C++ a poskytuje bohaté možnosti pro vývoj multiplatformních aplikací s moderním vzhledem. Disponuje širokou dokumentací, ve které jsou jednotlivé objekty detailně popsány, ale také demonstruje jejich použití v kódu. Komponenty se dají také stylizovat a to jednoduše pomocí kaskádových stylů, které mají stejnou syntaxi jako známé CSS. Lákadlem může být i možnost využití Qt Designeru, což je vizuální nástroj, který umožňuje vytvořit grafické rozhraní bez nutnosti psát jakýkoliv kód. Vzhledem k tomu, že pro projekt stačí jednoduché rozhraní, které obsahuje pár komponent, jsem se rozhodl Designer nevyužít a postupovat čistě programově.

2.4 Ostatní použité knihovny

Samozřejmě, že v projektu je i několik dílčích knihoven, které jsou nezbytné pro dnešní tvorbu jakýchkoliv skriptů. Za zmínku například stojí:

- shutil – Modul ze standardní knihovny Pythonu, který poskytuje funkce pro práci se souborovým systémem, jako je kopírování, přesouvání a mazání souborů a adresářů.
- tqdm – Knihovna pro zobrazování pokroku při iteraci (progress bar). Velmi užitečná při dlouhotrvajících operacích, jako je trénování modelů nebo zpracování dat.
- warnings – Modul pro správu varování v Pythonu. Využíváme s funkcí filterwarnings, protože Keras těchto varování vypisuje spoustu a v terminálu se pak nelze orientovat.
- numpy – Základní knihovna pro vědecké výpočty v Pythonu. Poskytuje podporu pro vícerozměrná pole, matematické funkce, lineární algebru a mnoho dalších numerických operací.
- json – Modul pro práci s formátem JSON, běžně používaným pro ukládání a přenos strukturovaných dat.
- seaborn – Vyšší nadstavba knihovny matplotlib pro vizualizaci dat. Umožňuje snadno vytvářet statistické grafy s atraktivním a informativním designem.
- matplotlib.pyplot – Modul knihovny matplotlib poskytující rozhraní podobné MATLABu pro tvorbu grafů a vizualizací v Pythonu.
- argparse – Modul standardní knihovny pro zpracování argumentů z příkazové řádky. Umožňuje definovat očekávané parametry a automaticky generuje nápovědu.

3 Implementace

Projekt byl realizován jako aplikace v jazyce Python. Skládá se z několika samostatných skriptů, které mají přesně vymezené funkce a mohou být využity i samy o sobě, bez nutnosti grafického rozhraní.

3.1 Struktura projektu

Projekt je uspořádán ve složce `diseaseDetector`, která obsahuje několik důležitých podsložek a souborů. Virtuální prostředí s potřebnými Python balíčky je uloženo ve složce `env`. Surová data jsou uložena ve složce `PlantVillage-Dataset`. Předzpracovaná a uspořádaná data, připravená k tréninku a validaci modelu, se nacházejí ve složce `public`, která obsahuje podsložky `train` a `val` pro trénovací a validační sady, a také `icons` pro grafické ikony používané v `GUI`. Dále obsahuje adresáře `models` a `histories`. Zde se nachází model, který byl mnou již natrénován a také historie jeho trénování po dobu epoch.

Zdrojové kódy projektu jsou ve složce `src`, kde najdeme skripty `gui.py` pro grafické uživatelské rozhraní, `main.py` jako hlavní spouštěcí skript řídící trénování a vyhodnocování modelu, `parseDataset.py` pro zpracování a organizaci datasetu do struktury použitelných trénovacích a validačních adresářů, a soubor `stylesheet.css`, který definuje vzhled `GUI`. Projekt dále obsahuje konfigurační soubory `.gitignore`, určující, které soubory Git ignoruje, a `requirements.txt`, obsahující seznam všech potřebných `Python` knihoven a jejich verzí.

```
diseaseDetector
├── env
├── PlantVillage-Dataset
├── public
│   ├── icons
│   ├── train
│   ├── val
│   ├── models
│   └── histories
├── src
│   ├── gui.py
│   ├── main.py
│   ├── parseDataset.py
│   └── stylesheet.css
├── .gitignore
└── requirements.txt
```

3.2 Příprava datové sady

Příprava datasetu představuje klíčovou fázi celého projektu, protože kvalita a struktura dat přímo ovlivňuje úspěšnost trénování a následné schopnosti modelu detekovat onemocnění rostlin. V projektu vycházíme ze surového datasetu `PlantVillage-Dataset`, který obsahuje rozsáhlou sbírku obrázků listů rostlin s různými typy onemocnění a také zdravých exemplářů. Tento dataset je stažený ve své původní formě a není přímo vhodný pro trénink modelu – je tedy potřeba jej upravit a přeuspořádat.

Hlavní náplní této fáze je tedy skript `parseDataset.py`, který slouží k extrakci, transformaci a přípravě dat do struktury vhodné pro model trénovaný v knihovně `Keras`. Tento skript provádí několik důležitých úkolů:

- Výběr podporovaných tříd – z celkového datasetu jsou zvoleny jen rostliny, pro které existují jak zdravé, tak nemocné vzorky. Zároveň dochází ke sjednocení názvů.

```
selectedProducts = {
    "Apple": "Apple",
    "Cherry_(including_sour)": "Cherry",
    "Corn_(maize)": "Corn",
```

```
    "Grape": "Grape",
    "Peach": "Peach",
    "Pepper,_bell": "Pepper",
    "Potato": "Potato",
    "Strawberry": "Strawberry",
    "Tomato": "Tomato"
}
```

- Vytvoření cílové adresářové struktury – jsou vytvořeny podsložky pro trénovací a validační sadu, rozdělené dále na zdravé a nemocné vzorky.

```
for dir in dirs:
    for subdir in subdirs:
        Path(f"{target}/{dir}/{subdir}").mkdir(parents=True, exist_ok=True)
```

- Filtrování a štítkování – pro každou složku v datasetu se určí, zda vzorek patří mezi zdravé nebo nemocné, a jestli je daný produkt v seznamu podporovaných.

```
for dir in tqdm(classes, "Creating dataset, please wait..."):
    product, disease = dir.split("___")
    if (product not in selectedProducts):
        continue
    label = "healthy" if disease.strip() == "healthy" else "diseased"
```

- Zamíchání a rozdělení dat – obrázky jsou náhodně zamíchány a rozděleny v poměru 80:20 do složek `train` a `val`.

```
images = os.listdir(os.path.join(source, dir))
random.shuffle(images)

splittedFiles = {
    "train" : images[:int(len(images) * 0.8)],
    "val" : images[int(len(images) * 0.8):]
}
```

- Přejmenování a kopírování obrázků – každý obrázek je přejmenován do konzistentního formátu a uložen do cílové složky podle svého rozdělení a štítku.

```
index = 0
for split, files in splittedFiles.items():
    for name in files:
        src = os.path.join(source, dir, name)
        suffix = os.path.splitext(name)[1]
        finalName = f"{selectedProducts[product]}_{disease}_{index}_{suffix}"
        dst = os.path.join(target, split, label, finalName)
        shutil.copyfile(src, dst)
        index += 1
```

3.3 Trénování a vyhodnocení modelu

Po předzpracování datové sady následuje klíčová fáze a to samotné trénování modelu a jeho následná evaluace. Cílem této fáze je naučit model rozpoznávat, zda se jedná o zdravou, či nemocnou rostlinu a to na základě vstupních obrázků. Je nutné dbát na to, aby byl výsledný model dobře generalizovatelný i na dříve neviděná data a mohl tak efektivně predikovat. Jelikož datová sada disponuje nevyváženými daty mezi třídami `healthy` a `diseased` je využita strategie vyvážení tříd pro minimalizaci zkreslení. Důraz je kladen nejen na správnou volbu metrik, ale i na techniky jako je regulace učení, dropout, L2 regulace a další mechanismy omezující přeučení. Pro optimalizaci trénovacího procesu jsou nasazeny různé `callbacky`, které automaticky reagují na vývoj metrik a řídí zastavení učení či úpravu učícího poměru.

Evaluace modelu zahrnuje výpočet přesnosti, preciznosti, ale zejména F1 skóre, které je u nevyvážených tříd považováno za nejvhodnější metriku. Model je zároveň testován pomocí konfuzní matice, která nabízí přehled o typech chyb, které model nejčastěji provádí. Dále je vizualizován průběh učení ve formě grafů vývoje metrik napříč epochami.

Následující části podrobně popisují jednotlivé kroky trénování a evaluace, včetně výběru architektury, nastavení parametrů, volby metrik a implementace konkrétních mechanismů řízení učení, které se vyskytují ve skriptu `main.py`.

3.3.1 Předzpracování dat

Pro trénování i validaci je využita knihovna `ImageDataGenerator`, která zajišťuje načtení dat z adresářové struktury a jejich normalizaci. Rozlišení vstupních obrázků je zmenšeno na 64×64 pixelů kvůli snížení výpočetních nároků. Augmentace nebyla použita, protože jejím vlivem došlo k zhoršení výsledků modelu.

```
trainData = preprocessing.image.ImageDataGenerator(rescale = 1./255)
valData = preprocessing.image.ImageDataGenerator(rescale=1./255)

trainGen = trainData.flow_from_directory(
    "../public/train",
    target_size=(64, 64),
    batch_size=64,
    class_mode="categorical"
)
```

3.3.2 Vyvážení tříd

Dataset obsahuje nerovnoměrně zastoupené třídy, což může ovlivnit kvalitu klasifikace. Z toho důvodu je při trénování využita váha tříd `class_weight`, vypočtená na základě četnosti výskytu každé třídy v trénovací množině.

```
classWeight = dict(zip(
    np.unique(trainGen.classes),
    class_weight.compute_class_weight(
        class_weight="balanced",
        classes=np.unique(trainGen.classes),
        y=trainGen.classes
```

```
    )
))
```

3.3.3 Architektura modelu

Model je postaven jako sekvenční síť s třemi konvolučními bloky. Každý blok se skládá z konvoluční vrstvy, normalizace, aktivační funkce ReLU, pooling vrstvy a dropout vrstvy. Plně propojené vrstvy využívají také dropout a normalizaci. Celý model používá L2 regularizaci.

```
model = models.Sequential([
    layers.Conv2D(32, (3, 3), input_shape=(64, 64, 3),
                  padding="same", kernel_regularizer=regularizers.l2(0.001)),
    layers.BatchNormalization(),
    layers.Activation("relu"),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Dropout(0.3),
    ...
    layers.Dense(64),
    layers.BatchNormalization(),
    layers.Activation("relu"),
    layers.Dropout(0.4),
    layers.Dense(trainGen.num_classes, activation="softmax")
])
```

3.3.4 Kompilace a metriky

Model je kompilován s Adam optimalizátorem a ztrátovou funkcí `categorical_crossentropy`. Mezi sledované metriky patří `accuracy`, `precision`, `recall` a `F1-score`. Právě vážené `F1-score` je klíčovým kritériem pro zastavení trénování a uložení nejlepší verze modelu.

```
model.compile(
    optimizer=optimizers.Adam(learning_rate=0.0001),
    loss="categorical_crossentropy",
    metrics=[
        "Accuracy", "Precision", "Recall",
        metrics.F1Score(average="weighted", name="F1Score")
    ]
)
```

3.3.5 Trénování modelu

Trénování probíhá po dobu až 20 epoch s využitím několika callbacků. Mezi ně patří `EarlyStopping`, `ModelCheckpoint` a `ReduceLROnPlateau`. Díky těmto callbackům lze automaticky ukládat nejlepší model a optimalizovat proces učení.

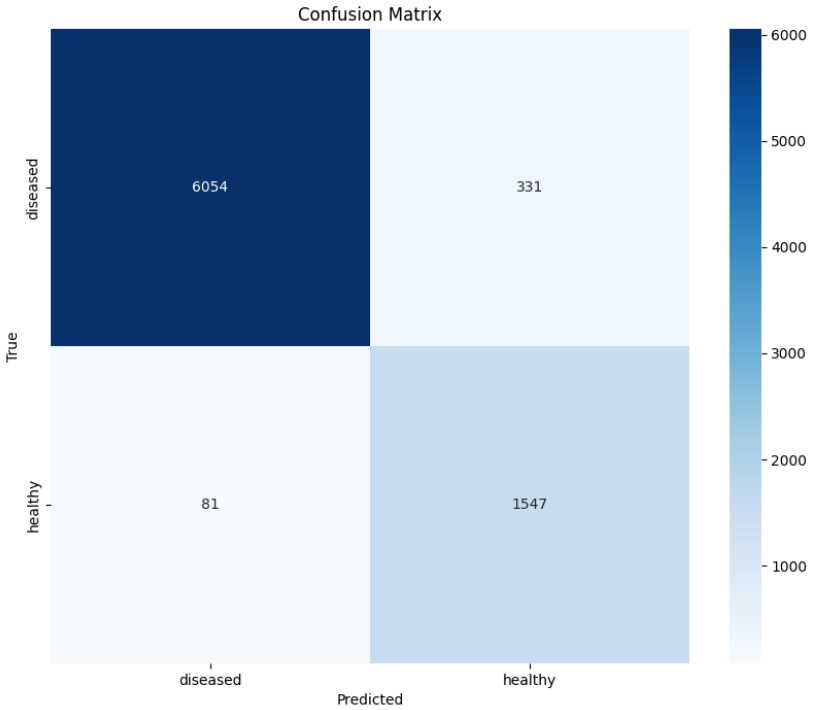
```
history = model.fit(
    trainGen, validation_data=valGen, epochs=20,
    class_weight=classWeight,
```

```
    callbacks=[
        callbacks.EarlyStopping(
            patience=5, restore_best_weights=True,
            monitor="val_F1Score", verbose=1
        ),
        callbacks.ModelCheckpoint(
            "best_" + modelName, monitor="val_F1Score",
            mode="max", save_best_only=True, verbose=1
        ),
        callbacks.ReduceLROnPlateau(
            monitor="val_F1Score", factor=0.5,
            patience=3, verbose=1, min_lr=1e-6
        )
    ]
)
```

3.3.6 Vyhodnocení modelu

Vyhodnocení se provádí na validační množině. Vypisuje se klasifikační report a zobrazuje se konfuzní matice pomocí knihovny `seaborn`.

```
predictions = model.predict(valGen)
predictedClasses = np.argmax(predictions, axis=1)
trueClasses = valGen.classes
classLabels = list(valGen.class_indices.keys())
print(classification_report(trueClasses, predictedClasses,
      target_names = classLabels))
sns.heatmap(confusion_matrix(trueClasses, predictedClasses), annot=True)
```

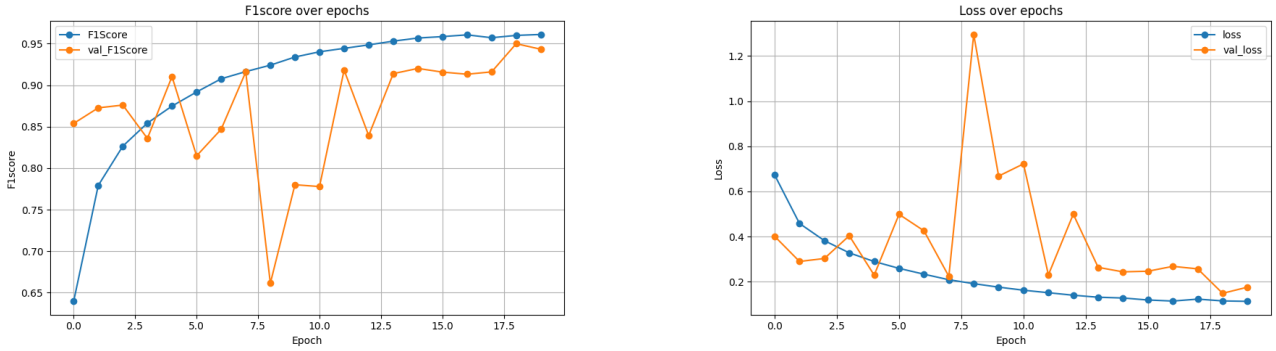


Obrázek 1: Matice záměn natrénovaného modelu

3.3.7 Vizualizace průběhu učení

Průběh učení je ukládán do `JSON` souboru a následně lze pomocí funkce `printHistory` zobrazit vývoj metrik v čase. Pokud model mění učící poměr, je zobrazen i jeho vývoj.

```
with open("pathToFile", "r") as f:
    history = json.load(f)

plt.plot(history["F1Score"])
plt.plot(history["val_F1Score"])
plt.title("F1-Score over epochs")
```



Obrázek 2: Grafy F1 skóre a ztrát v průběhu vývoje modelu

3.3.8 Spouštění skriptu

Skript lze spustit pomocí argumentů z příkazové řádky:

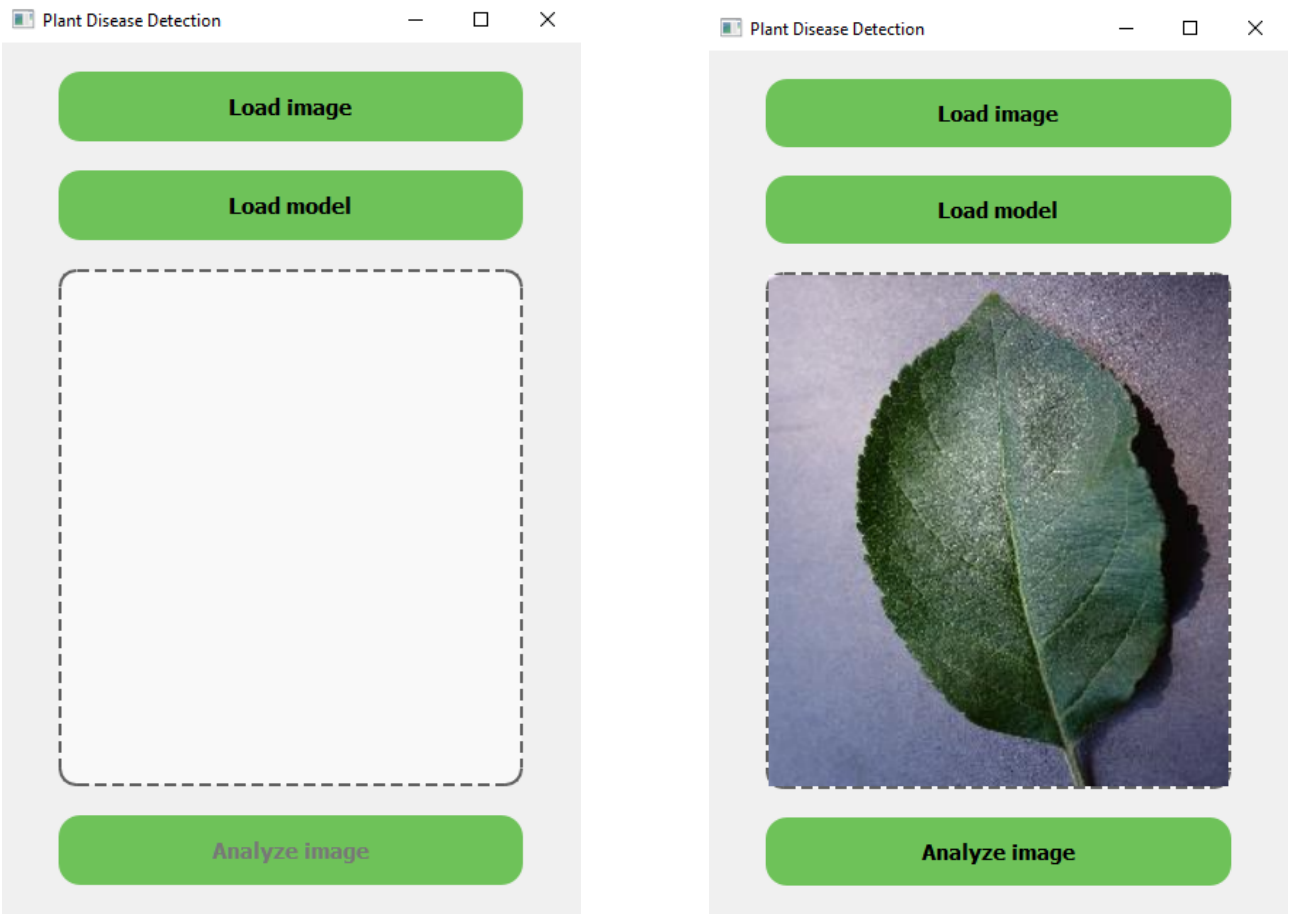
```
python3 ./main.py -t <model_name.keras>          # trénování
python3 ./main.py -e <path_to_model.keras>       # evaluace
python3 ./main.py -h <path_to_history.json>      # vizualizace průběhu
python3 ./main.py -u                             # vypíše pomoc, jak skript spustit
```

3.4 Uživatelské rozhraní

Pro účely praktického využití bylo vytvořeno jednoduché grafické uživatelské rozhraní pomocí frameworku `PyQt5`, který umožňuje snadnou práci s vytrénovaným modelem. Uživatel v něm může:

- Načíst model neuronové sítě ve formátu `.h5` či `.keras`.
- Nahrát libovolný obrázek listu ve formátu `jpg, jpeg, png`.
- Spustit klasifikaci a přehledně zobrazit výsledek uživateli.

Rozhraní má fixní velikost 400×600 pixelů a je navrženo tak, aby bylo automaticky vycentrováno na obrazovce.



Obrázek 3: Podoba grafického rozhraní před a po načtení parametrů

3.4.1 Vzhled a ovládání

Aplikace obsahuje tři hlavní tlačítka – `Load image`, `Load model` a `Analzye image`. První dvě slouží k načtení vstupních dat a modelu. Třetí je aktivní pouze v případě, že jsou obě předchozí položky načtené. Krom těchto tlačítek je implementovaná také oblast, do které je následně načten obrázek uživatelem. Odezva aplikace je zajištěna pomocí pop-up dialogů.

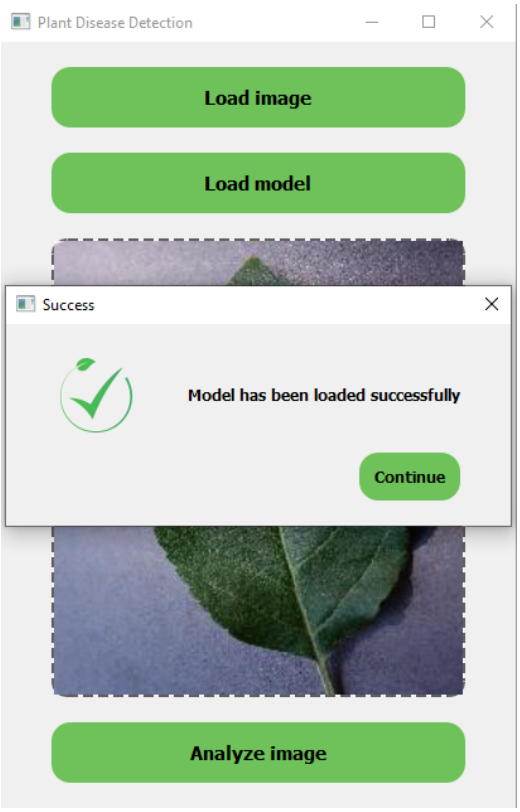
```
layout = QVBoxLayout(centralWidget)
layout.setContentsMargins(40, 20, 40, 20)

self.buttonLoad = QPushButton("Load image", self)
self.buttonLoadModel = QPushButton("Load model", self)
self.buttonAnalyze = QPushButton("Analyze image", self)
self.imageLabel = QLabel(self)

layout.addWidget(self.buttonLoad)
layout.addWidget(self.buttonLoadModel)
layout.addWidget(self.imageLabel)
layout.addWidget(self.buttonAnalyze)
```

3.4.2 Analýza a načtení modelu

Model i obrázek jsou načítány pomocí standardních dialogových oken. Po jejich načtení je aktivní tlačítko pro spuštění analýzy.

```
def loadModel(self):
    fileName, _ = QFileDialog.getOpenFileName(
        self, "Open Model", self.dir, "Keras files (*.h5 *.keras)"
    )
    if fileName:
        self.model = fileName
        alert = CustomDialog("Success", "Model has been loaded successfully",
                self)
def loadImage(self):
    fileName, _ = QFileDialog.getOpenFileName(self, "Open Image", self.dir,
                  "Images Files (*.png *.jpg *.jpeg)")
    if (fileName):
        pixmap = QPixmap(fileName)
        self.imageLabel.setPixmap(pixmap.scaled(self.imageLabel.size(),
        Qt.IgnoreAspectRatio, Qt.SmoothTransformation))
```

V případě úspěšného nahrání modelu je uživatel obeznámen s touto skutečností prostřednictvím pop-up okna, které je vytvořeno jako třída `CustomDialog(QDialog)`. Po načtení obrázku je předdefinovaná oblast pro obrázek příslušným obrázkem vyplněna viz. obrázek 3.



Obrázek 4: Pop-up okno se zpětnou vazbou uživateli

3.4.3 Zpracování obrázku

Po kliknutí na `Analyze image` dojde k načtení modelu pomocí funkce `load_model`, která nabízí uživatelům jednoduchou možnost, jak nahrát jakýkoliv před-trénovaný model ze svého disku do různých rozhraní. Má pouze jeden parametr a to je cesta k příslušnému modelu. Stejně tak i samotný obrázek, jenž uživatel do aplikace vložil, musí být ještě zpracován. Funkce `load_img` umožňuje jednoduše tento soubor znormalizovat na požadovaný rozměr, což je v našem případě 64×64 pixelů.

```
model = models.load_model(self.model)
file = preprocessing.image.load_img(self.filePath, target_size=(64, 64))
fileArray = preprocessing.image.img_to_array(file) / 255.0
fileArray = np.expand_dims(fileArray, axis=0)
prediction = np.argmax(model.predict(fileArray), axis=1)
```

Následně je pak provedena klasifikace pomocí ternárního výrazu a v závislosti na jeho výsledku je uživateli zobrazen příslušný dialog.

```
self.result = True if (prediction[0]) else False
if self.result:
    dialog = CustomDialog("Analysis Result",
    "Congratulations, your plant is healthy!", self)
else:
    dialog = CustomDialog("Analysis Result", "Unfortunately your plant is diseased!",
    self, "../public/icons/notOk.png")
dialog.exec_()
```

3.5 Zprovoznění projektu

Pro spuštění projektu je doporučeno vytvořit si samostatné virtuální prostředí, do kterého se následně nainstalují všechny potřebné knihovny. Aktivace virtuálního prostředí eliminuje možné konflikty ohledně cest k jednotlivým knihovnám.

- Otevřete terminál ve složce s projektem.
- Vytvořte nové virtuální prostředí (například pomocí `venv`):

  ```
  python -m venv env
  ```

- Aktivujte virtuální prostředí:
  - Na Windows:

    ```
    C:\> <venv>\Scripts\activate.bat
    ```

  - Na Linuxu nebo macOS:

    ```
    $ source <venv>/bin/activate
    ```

- Nainstalujte všechny potřebné závislosti ze souboru `requirements.txt`:

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

- Spusťte aplikaci:

  ```
  python ./gui.py
  ```

  Po spuštění se otevře grafické rozhraní, kde lze načíst model a obrázek listu, a provést analýzu. Alternativně pak lze i spouštět skript `main.py`, dle popisu v 3.3.8.

4 Závěr a možné zlepšení

V této práci byl úspěšně navržen a implementován klasifikační model pro rozpoznání zdravotního stavu rostlin na základě analýzy obrazových dat. Model byl integrován do jednoduchého grafického uživatelského rozhraní, které umožňuje uživatelům snadné načtení modelu a obrázku, provedení klasifikace a zobrazení výsledku. Testování ukázalo, že aplikace je schopna spolehlivě rozlišovat mezi zdravými a nemocnými rostlinami.

Přestože je současná verze aplikace funkční, existuje několik oblastí, kde by bylo možné projekt dále rozvinout a zlepšit:

- **Zvýšení přesnosti modelu:** Využití rozsáhlejších a rozmanitějších datasetů, případně zapojení technik transfer learningu s moderními architekturami hlubokých neuronových sítí by mohlo významně zvýšit kvalitu klasifikace.
- **Rozšíření počtu klasifikačních tříd:** Místo jednoduchého rozlišení zdravý/ nemocný by bylo možné model rozšířit o detekci konkrétních typů chorob, což by mohlo pomoci při cílenější léčbě.
- **Vylepšení uživatelského rozhraní:** Přidání dalších funkcionalit, jako je možnost ukládání výsledků nebo podpora mobilních zařízení, by zlepšilo použitelnost aplikace.
- **Optimalizace rychlosti klasifikace:** Nasazení modelu na výkonnější hardware nebo jeho optimalizace pro běh na nízkonákladových zařízeních by umožnilo rychlejší zpracování a širší využití.