Документација за изработен проект по предметот Вештачка интелигенција под наслов

Модел за препознавање на болести кај растенија

Изработила:

Елена Николовска 225004

Ментор:

Андреа Кулаков

Содржина

Вовед

Болестите кај растенијата претставуваат голем проблем во земјоделието. Тие влијаат врз приносите и квалитетот на земјоделските култури како и економијата на земјоделските производи. Појавата на овие болести може да биде предизивикана од различни фактори како бактерии, габи, вируси но и неповолни климатски промени. Доколку овие болести не се идентификуваат и третираат навремено можат да предизвикаат значителни загуби во производството како и намалување на квалитетот на прозиводите и зголемување на употребата на хемиски пестициди.

Традиционалниот начин за препознавање на болести кај растенијата често вклучува експерти од областа на земјоделието кои донесуваат заклучоци врз основа на визуелна инспекција на заболеното растение. Меѓутоа, овој начин е често скап но и не толку ефикасен, поради тоа што не секој прозиводител на култури може да си дозволи мислење и надзор од стручно лице.

Со развојот на технологијата за машинско учење и машинска визија, автоматизираните системи за препознавање на болести кај растенија стануваат популарни но и поефикасни. Со нивна употреба се доаѓа до побрза но и попрецизина детекција на болест кај заболените растенија, со што можат навремено да се превземат мерки за нивна заштита како и спречување на понатамошно проширување.

Овој проект развива модел за препознавање на болести кај растенија преку индетификација на различни типови на заболувања користејќи слики од растенија.

Бактериско заболување кај пиперка

Прашкаста мувла кај цреша

Бактериска патогена болест кај јагода

CNN архитектура

Convolutional Neural Networks (CNNs) се напреден тип на вештачки невронски мрежи дизајнирани за обработка и анализа на слики, препознавање на слики, класификација и детекција на објекти. Тие се составени од повеќе слоеви кои автоматски учат карактеристики од сликите без потреба од рачно извлекување на атрибути.

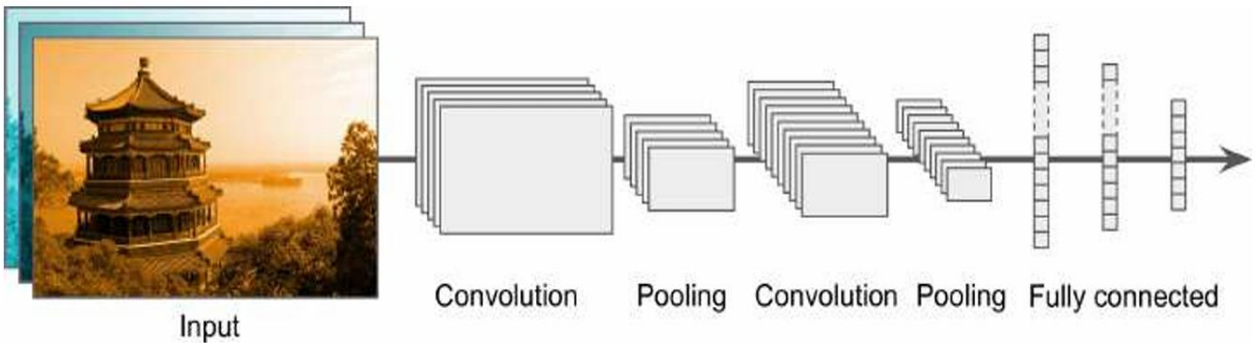
Типичните архитектури на CNN редат неколку конволутивни (convolutional) слоеви (секој од нив следен од ReLU слој), потоа следи пулинг (pooling) слој па повторно конволутивни (convolutional) слоеви (+ ReLU слој) па пулинг (pooling) слој и така натаму.

Сликата станува се помала и помала како што напредува низ мрежата но исто така станува со повеќе мапи на карактеристики (feature maps).

Конволутивните слоеви користат конволутивни филтри (kernels) за екстракција на важни карактеристики од сликите, како што се рабови, текстури и форми.

Пулинг слоевите ја намалуваат димензионалноста на сликата (downsampling) со што се зачувуваат најважните карактеристики, а се намалува пресметковната комплексност.

Со текот на годините се развивале различни варијанти на оваа фундаментална архитектура како на пример LeNet, AlexNet, VGG, ResNet и EfficientNet што довело до неверојатен напредок во областа.



VGGNet архитектура

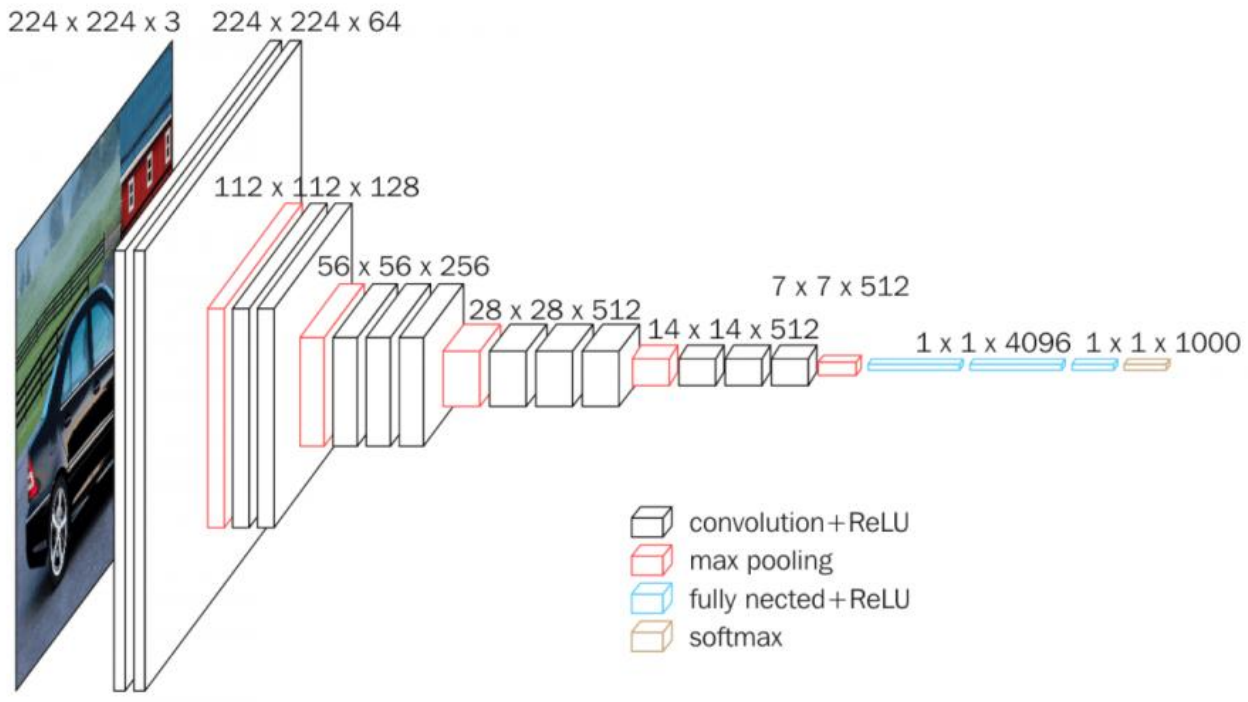
VGGNet е популарна CNN архитектура предложена од Visual Geometry Group (VGG) на Универзитетот Оксфорд. Таа е позната по својата едноставност и одлични резултати во класификација на слики.

Архитектурата на VGGNet е хиерархиски организирана, каде што секвенци од 2 или 3 конволутивни слоеви (со филтри од 3×3) се следени од пулинг слој. Овој образец се повторува неколку пати, што резултира со вкупно 16 конволутивни слоеви (во VGG-16). По овие слоеви, архитектурата завршува со целосно поврзана (dense) мрежа, составена од два скриени слоеви и еден излезен слој.

Користењето на мали 3×3 филтри во сите конволутивни слоеви овозможува подобро учење на карактеристиките и намалување на бројот на параметри во однос на поголеми филтри.

За воведување нелинеарност и подобра стабилност, по секој конволутивен слој се применува Rectified Linear Unit (ReLU) активациска функција. На крајниот слој, Softmax функцијата пресметува веројатности за секоја класа, овозможувајќи точна класификација на сликите.

Влезната слика во VGGNet мора да биде со големина 224×224 пиксели и да има 3 RGB канали.



Зголемување на податоците (Data augmentation)

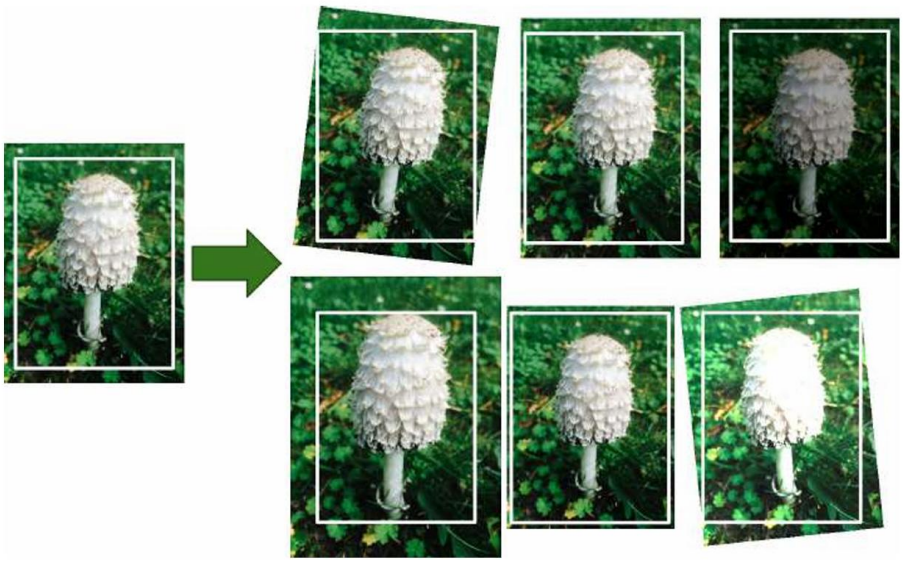
Зголемување на податоците (Data augmentation) вештачки ја зголемува големината на тренирачкото множество со генерирање на многу реалистички варијанти од секоја тренирачка инстанца. Ова намалува пренавикнување со што ја прави техника на регулација.

Генерираните примери треба да бидат што е можно пореални, идеално на дадена слика од аугментираното тренирачко множество, човек не треба да биде способен да погоди дали сликата е агументирана или пак не.

Покрај тоа едноставно додавање на бел шум нема да биде од помош, бидејќи измените треба да можат да се научат (белиот шум не може).

Измените кои ги применив во проектот се следниве:

```
rotation_range=30 - ротација на случаен агол во опсег од 30 степени.
width_shift_range=0.2 – хоризонтално поместување за максиумум 20% од ширината
height_shift_range=0.2 – вертикално поместување за максиумум 20% од висината
shear_range=0.2 – деформирање на слика под агол
zoom_range=0.2 – зумирање на сликата за максимум 20%
horizontal_flip=True – хоризонтално превртување
```



Пример за data augmentation

Модел 1

За распознавање на болестите кај растенијата имам направено два модела.

Во двата модела се користат истите data augmentation што се објаснети погоре. Пред да започнеме со тренирањето на моделот, на сликите од тренирачкото множество и множеството за валидација правиме нормализација на вредноста на пикселите во опсег од [0,1]. Исто така сликите ги нормализирме во димензии 224 x 224 кои се потребни за VGGNet архитектурата бидејќи го користиме овој модел.

Во првиот модел ги користиме сите веќе тренирани слоеви од моделот, што значи дека ги замрзнуваме слоевите на базниот модел и ги тренираме само новите што ги додаваме. Не сакаме да ги тренираме сите слоеви на моделот, бидејќи тие се веќе претходно тренирани за да извлекуваат карактеристики од слики (на пример рабови, текстури и слично).

Замрзнувањето на слоевите во базниот модел значи дека тие ќе останат фиксни и нема да се променуваат за време на тренирањето, што ќе го забрза процесот и ќе го насочи фокусот на тренирање на новите додадени слоеви, наместо на веќе тренираните. Тоа се прави со следниов код:

```
for layer in base_model.layers:
    layer.trainable = False
```

Следниот чекор е подготовка на излезот од конволутивните слоеви за да влезе во густите поврзаните слоеви (Dense layers) кои следат.

Додаваме dense layer со 512 неврони. Секој од овие неврони има своја тежина што ќе се тренира за време на тренирањето. Овде користиме и ReLU (Rectified Linear Unit) што ја заменува негативната вредност на излезот со нула и ја остава позитивната вредност непроменета. Ова помага во избегнување на проблемот со исчезнување на градиентот.

За да спречиме пренавикнување на моделот (overfitting) случајно исклучуваме 50% од невроните за време на тренирањето.

Во последниот dense слој користиме 24 неврони за 24 класи во моделот и мултикласна класификација, Softmax која обезбедува веројатности за секоја класа и овозможува одлучување во која класа е најверојатно дека ќе се класифицира сликата.

Тоа се прави со следниов код:

```
model = models.Sequential([
    base_model,
    layers.Flatten(),
    layers.Dense(512, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(24, activation='softmax')
])
```

Резултати од тренирањето и тестирањето:

Epoch 1/20 accuracy: 0.4354 - loss: 2.0747 - val_accuracy: 0.7227 - val_loss: 0.8243

Epoch 2/20 accuracy: 0.6749 - loss: 0.9706 - val_accuracy: 0.8044 - val_loss: 0.6152

Epoch 3/20 accuracy: 0.7206 - loss: 0.8283 - val_accuracy: 0.8129 - val_loss: 0.5223

Epoch 4/20 accuracy: 0.7340 - loss: 0.7879 - val_accuracy: 0.8385 - val_loss: 0.4567

Epoch 5/20 accuracy: 0.7384 - loss: 0.7622 - val_accuracy: 0.8329 - val_loss: 0.4753 (знаци на overfitting)

Epoch 6/20 accuracy: 0.7441 - loss: 0.7449 - val_accuracy: 0.8394 - val_loss: 0.4673

Epoch 7/20 accuracy: 0.7548 - loss: 0.7214 - val_accuracy: 0.8204 - val_loss: 0.5241

Epoch 8/20 accuracy: 0.7658 - loss: 0.6859 - val_accuracy: 0.8440 - val_loss: 0.4382 (се враќа)

Epoch 9/20 accuracy: 0.7741 - loss: 0.6671 - val_accuracy: 0.8502 - val_loss: 0.4069

Epoch 10/20 accuracy: 0.7809 - loss: 0.6468 - val_accuracy: 0.8444 - val_loss: 0.4487 (знаци на overfitting)

Epoch 11/20 accuracy: 0.7808 - loss: 0.6532 - val_accuracy: 0.8527 - val_loss: 0.4163

Epoch 12/20 accuracy: 0.7837 - loss: 0.6280 - val_accuracy: 0.8583 - val_loss: 0.4017

Epoch 13/20 accuracy: 0.7956 - loss: 0.6051 - val_accuracy: 0.8496 - val_loss: 0.4020

Epoch 14/20 accuracy: 0.7937 - loss: 0.6096 - val_accuracy: 0.8525 - val_loss: 0.3942

Epoch 15/20 accuracy: 0.7956 - loss: 0.5922 - val_accuracy: 0.8667 - val_loss: 0.3771

Epoch 16/20 accuracy: 0.8002 - loss: 0.5835 - val_accuracy: 0.8646 - val_loss: 0.3645

Epoch 17/20 accuracy: 0.8006 - loss: 0.5929 - val_accuracy: 0.8775 - val_loss: 0.3432

Epoch 18/20 accuracy: 0.8032 - loss: 0.5826 - val_accuracy: 0.8727 - val_loss: 0.3547

Epoch 19/20 accuracy: 0.8011 - loss: 0.5814 - val_accuracy: 0.8652 - val_loss: 0.3553

Epoch 20/20 accuracy: 0.8068 - loss: 0.5666 - val_accuracy: 0.8675 - val_loss: 0.3738

Test Accuracy: 0.8792

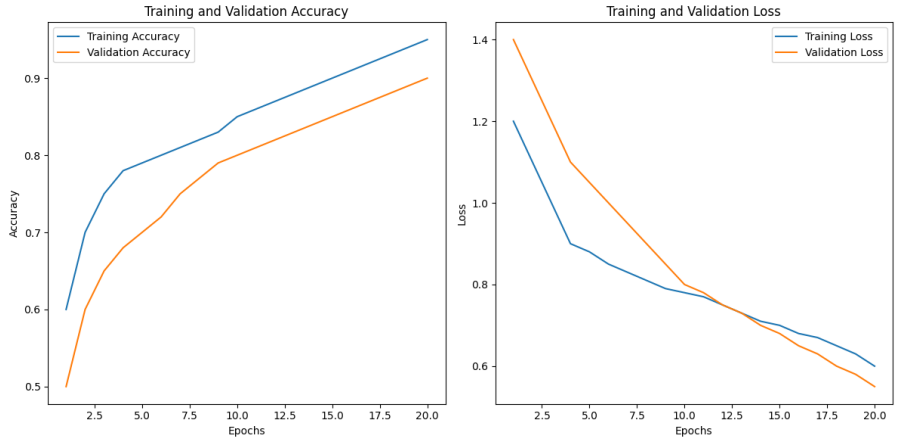
Точноста на тренинг множеството се зголемува од 43.5% на 80.7%, што покажува дека моделот успешно учи од податоците.

Губитокот (loss) се намалува од 2.07 на 0.57, што значи дека предвидувањата стануваат попрецизни.

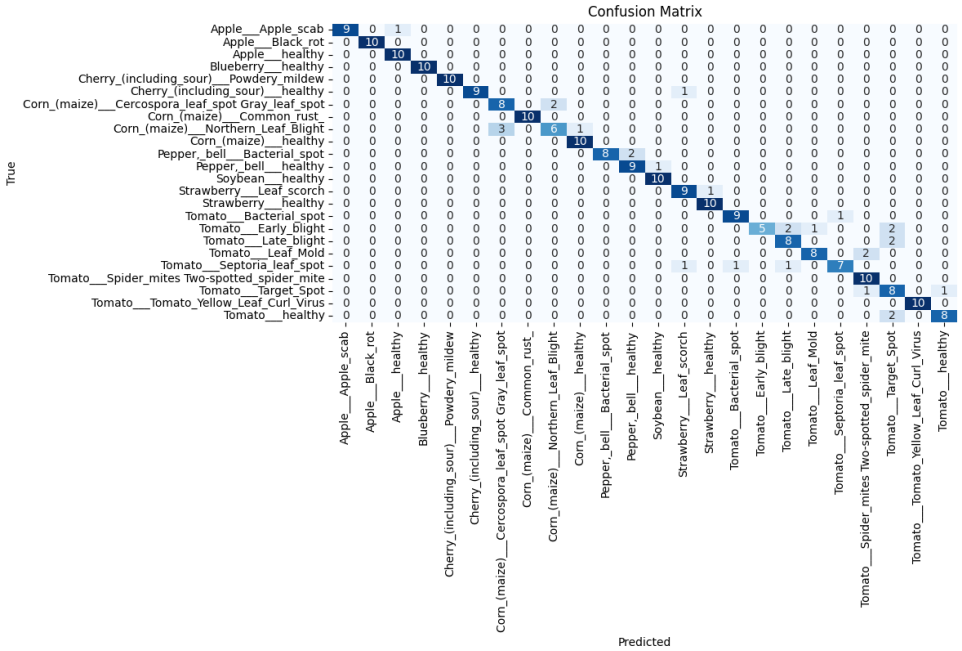
Точноста на валидацијата се подобрува од 72.27% на 86.75%, што покажува дека моделот добро генерализира на нови податоци.

Губитокот на валидацијата се намалува од 0.82 на 0.37, но има забележливи флуктуации, што може да покаже на overfitting.

Веќе во Epoch 5 и Epoch 10, моделот покажува знаци на overfitting – точноста на тренинг множеството продолжува да расте, но точноста на валидацијата не се зголемува пропорционално или дури се намалува.



Графици на перформансите на моделот при тренирање



Матрица на конфузија

Модел 2

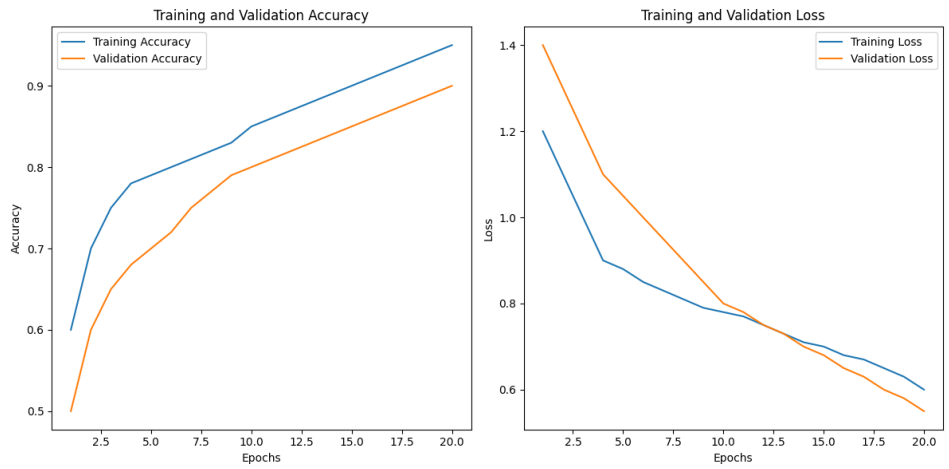
Бидејќи во модел 1 забележавме знаци на overfitting, за подобрување на моделот се направени некои измени. Една од измените е додавање на дополнителна промена во делот за data augmentation, со што се менува осветленоста на сликите односно тие ќе бидат малку потемни или малку посветли. Друга измена е тоа што за разлика од претходниот модел овде ги одмрзнуваме последните 2 слоја со што му овозможуваме на моделот да се прилагоди на нашите податоци а не да ги користи научените карактеристики на веќе трениранот модел.

Покрај овие измени се додадени и L2 регулација која на некој начин ги спречува тежините на невроните да станат премногу големи и помага во тоа моделот да не стане премногу сложен како и намалување на невроните од 512 на 256. Како последна промена е додадена стапката на учење (learning rate) која ќе придонесе моделот стабилно да се подобрува и бидејќи користиме претрениран модел ќе се погрижиме за тоа моделот да не ги расипува веќе научените тежини.

```
for layer in base_model.layers[:-2]:
    layer.trainable = False

model = models.Sequential([
    base_model,
    layers.Flatten(),
    layers.Dense(256, activation='relu',  kernel_regularizer=l2(0.001)),
    layers.Dropout(0.5),
    layers.Dense(24, activation='softmax')])


optimizer = tf.keras.optimizers.Adam(learning_rate=0.0001)
```



Графици на перформансите на моделот при тренирање

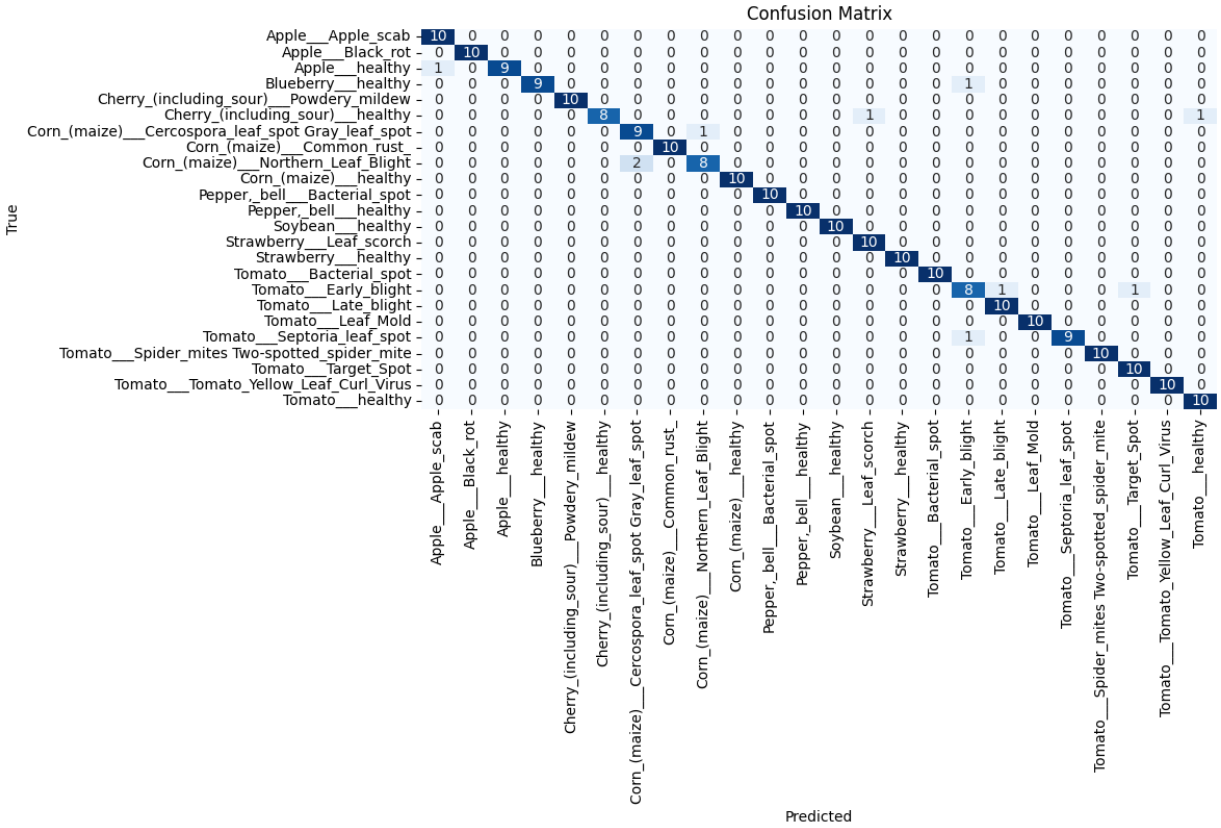
Резултати од тренирањето и тестирањето:

```
Epoch 1/20 accuracy: 0.6251 - loss: 1.6202 - val_accuracy: 0.8702 - val_loss: 0.6626 - learning_rate: 1.0000e-04
Epoch 2/20 accuracy: 0.9117 - loss: 0.5252 - val_accuracy: 0.9192 - val_loss: 0.4666 - learning_rate: 1.0000e-04
Epoch 3/20 accuracy: 0.9459 - loss: 0.3904 - val_accuracy: 0.9277 - val_loss: 0.4178 - learning_rate: 1.0000e-04
Epoch 4/20 accuracy: 0.9647 - loss: 0.3131 - val_accuracy: 0.9398 - val_loss: 0.3641 - learning_rate: 1.0000e-04
Epoch 5/20 accuracy: 0.9741 - loss: 0.2615 - val_accuracy: 0.9383 - val_loss: 0.3420 - learning_rate: 1.0000e-04
Epoch 6/20 accuracy: 0.9793 - loss: 0.2262 - val_accuracy: 0.9423 - val_loss: 0.3268 - learning_rate: 1.0000e-04
Epoch 7/20 accuracy: 0.9825 - loss: 0.1990 - val_accuracy: 0.9404 - val_loss: 0.3188 - learning_rate: 1.0000e-04
Epoch 8/20 accuracy: 0.9856 - loss: 0.1737 - val_accuracy: 0.9419 - val_loss: 0.2833 - learning_rate: 1.0000e-04
Epoch 9/20 accuracy: 0.9892 - loss: 0.1512 - val_accuracy: 0.9500 - val_loss: 0.2673 - learning_rate: 1.0000e-04
Epoch 10/20 accuracy: 0.9900 - loss: 0.1414 - val_accuracy: 0.9488 - val_loss: 0.2506 - learning_rate: 1.0000e-04
Epoch 11/20 accuracy: 0.9906 - loss: 0.1269 - val_accuracy: 0.9396 - val_loss: 0.2899 - learning_rate: 1.0000e-04
Epoch 12/20 accuracy: 0.9911 - loss: 0.1199 - val_accuracy: 0.9369 - val_loss: 0.2738 - learning_rate: 1.0000e-04
Epoch 13/20 accuracy: 0.9921 - loss: 0.1147 - val_accuracy: 0.9475 - val_loss: 0.2486 - learning_rate: 1.0000e-04
Epoch 14/20 accuracy: 0.9932 - loss: 0.1014 - val_accuracy: 0.9517 - val_loss: 0.2295 - learning_rate: 1.0000e-04
Epoch 15/20 accuracy: 0.9939 - loss: 0.0959 - val_accuracy: 0.9456 - val_loss: 0.2538 - learning_rate: 1.0000e-04
Epoch 16/20 accuracy: 0.9943 - loss: 0.0944 - val_accuracy: 0.9402 - val_loss: 0.2650 - learning_rate: 1.0000e-04
Epoch 17/20 accuracy: 0.9940 - loss: 0.0887 - val_accuracy: 0.9515 - val_loss: 0.2271 - learning_rate: 1.0000e-04
Epoch 18/20 accuracy: 0.9949 - loss: 0.0881 - val_accuracy: 0.9471 - val_loss: 0.2418 - learning_rate: 1.0000e-04
Epoch 19/20 accuracy: 0.9933 - loss: 0.0884 - val_accuracy: 0.9483 - val_loss: 0.2394 - learning_rate: 1.0000e-04
Epoch 20/20 accuracy: 0.9958 - loss: 0.0785 - val_accuracy: 0.9558 - val_loss: 0.2162 - learning_rate: 1.0000e-04
Test Accuracy: 0.9583
```

Почетната точност на тренинг сетот (Epoch 1) е 62.5%, а точноста на валидацијата е 87.02%. Во следните неколку епохи, точноста на тренинг сетот брзо се зголемува, достигнувајќи 99.58% по 20 епохи. Ова покажува дека моделот се тренира успешно и ја научил основната структура на податоците.

Додека точноста на тренинг множеството се зголемува многу брзо, точноста на валидацијата се стабилизира. Во некои епохи, како на пример во Epoch 11 и Epoch 16, забележано е мало намалување на точноста на валидацијата и поголемата стабилност. Тоа е знак на потенцијален overfitting, особено ако точноста на тестирањето потоа значително опадне во однос на точноста на тренирањето.

И тренинг загубата и загубата на валидацијата се намалуваат континуирано со секоја епоха. По 20 епохи, тренинг загубата е 0.0785 а загубата на валидацијата 0.2162, што покажува дека моделот ја минимизира грешката.



Матрица на конфузија

Заклучок

Препознавањето на болести кај растенијата игра значајна улога во земјоделството, бидејќи навремената детекција може да помогне во намалување на загубите и подобрување на квалитетот на производите. Традиционалните методи за препознавање, како визуелна инспекција од експерти во областа, иако ефикасни, често се скапи и не секогаш достапни за сите производители на земјоделски култури.

Резултатите од овој проект покажуваат дека со користење на машинско учење и машинска визија може да се постигне прецизно и автоматизирано препознавање на болести.

Точноста на моделот е висока, особено при тренирањето, но одредени варијации во валидацијата укажуваат на можен overfitting, што покажува дека има простор за дополнителни подобрувања или пак користење на некоја друга CNN наместо претрениранатa VGGNet16 што се користи во овој модел.

И покрај тоа, автоматизираните системи за детекција на болести имаат голем потенцијал за примена во земјоделството. Со понатамошно подобрување, ваквите модели можат да станат достапна и корисна алатка што ќе им помогне на земјоделците побрзо и полесно да ги заштитат своите растенија, да ја намалат употребата на хемиски препарати и да го подобрат приносот и квалитетот на истите.