Department of Computer Science
Technical University of Cluj-Napoca

Machine Learning Project

Laboratory activity

Plantling Recognition

Name: Hulea Andrei-Florin
Group: 30235
Email: andrei.hulea1@gmail.com

Teaching assistant: Roxana Ramona Szomiu
roxana.szomiu@cs.utcluj.ro

Contents

Chapter 1

Prezentarea problemei

1.1 Problema aleasa

In cadrul acestui assignment vom prezenta cum ne poate ajuta o reteaua neuronala convolutionala la clasificarea plantelor tinere prezente in setul de date. Acest lucru este greu de realizat cu ochiul liber chiar si daca avem cunostiinte in prealabil despre cum ar trebui sa arate, deoarece fiind abia rasarite, acestea seamana foarte mult intre ele.

Scopul acestui proiect este de a evidentia modul in care poate fi folosita o astfel de retea in viata de zi cu zi, cat si de a compara diferite metode de implementare a acesteia pentru a vedea care este cea mai eficienta.

Dataset-ul ales este un subset(v2) al dataset-ului Plant Seedlings Dataset si contine peste 5500 de imagini, care curpind 12 clase, fiecare clasa reprezentand un tip de planta.
0: 'Black grass'
1: 'Charlock'
2: 'Cleavers'
3: 'Common Chickweed'
4: 'Common wheat'
5: 'Fat hen'
6: 'Loose Silky-bent'
7: 'Maize'
8: 'Scentless Mayweed'
9: 'Sheperdś Purse'
10: 'Small-flowered Cranesbill'
11: 'Sugar beet

Fiecare clasa din dataset corespunde unui folder. Acestea au fost redenumite cu numerele corespunzatoare listei de mai sus pentru a usura implementarea. Link catre dataset
Ex: Small-flowered Cranesbill 10/10.png

Chapter 2

Detalii de implementare

2.1 Modelul ales

Modelul de deep learning ales pentru acest proiect este o retea neuronala convolutionala deoarece acestea au cea mai buna performanta in recunoasterea imaginilor.

Toate cele 5 modele incarca dataset-ul in acelasi mod si importeaza aceleasi librarii:

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from sklearn.model_selection import train_test_split
import cv2 as cv
from tensorflow.keras.layers import Dense, Activation, Flatten, Conv2D, MaxPooling2D
import os
import matplotlib.pyplot as plt
```

```
filenames = []
labels = []
classes = os.listdir('dataset')
for i in classes:
    imgs_name = os.listdir('E:/~SI/Plantling_Recognition2/dataset/' + i)
    for j in imgs_name:
        labels.append(int(i))
        filenames.append('E:/~SI/Plantling_Recognition2/dataset/' + i + '/' + j)

filenames = np.asarray(filenames)
labels = np.asarray(labels)
print(filenames)
print(labels)

print(filenames.shape)
print(labels.shape)

X_train_images, X_test_images, y_train_labels, y_test_labels = train_test_split(filenames,
                                                                                labels,
                                                                                test_size=0.2,
                                                                                random_state=45)

```

Parcurgem lista cu numele corespunzatoare directoarelor claselor si formam lista de label-uri si imagini. de train, dupa care aplicam functia train_test_split pentru a separa datele in date de antrenare si date de test. 20% din datele din dataset vor fi asignate pentru test. Parametrul random_state face ca la fiecare rulare sa nu se genereze valori noi random de fiecare data pentru train si test.

```
print("Number of training images: ", len(X_train_images))
print("Number of training labels: ", len(y_train_labels))
print("Number of testing images: ", len(X_test_images))
print("Number of testing labels: ", len(y_test_labels))
```

```
Number of training images:  4431
Number of training labels:  4431
Number of testing images:  1108
Number of testing labels:  1108
```

```
X_train = []
for i in range(len(X_train_images)):
    print(i)
    aux = cv.imread(X_train_images[i], cv.IMREAD_COLOR)
    aux = cv.resize(aux, (128, 128))
    X_train.append(aux)

X_test = []
for i in range(len(X_test_images)):
    print(i)
    aux = cv.imread(X_test_images[i], cv.IMREAD_COLOR)
    aux = cv.resize(aux, (128, 128))
    X_test.append(aux)
```

Urmatorul pas este sa parcurgem numele de fisiere si sa le adaugam in array-urile pentru test si train. Vom citi imaginile respective color cu ajutorul OpenCV si le vom redimensiona la 128x128 pixeli.

```
1
2  X_train = np.asarray(X_train)
3  X_test = np.asarray(X_test)
4  print("X_train.shape : " + str(X_train.shape))
5  print("X_test.shape : " + str(X_test.shape))
6  # plt.imshow(X_train[0])
7  # plt.show()
8  print("y_train shape: " + str(y_train_labels.shape))
9  print("a value from y_train: " + str(y_train_labels[1]))
10
```

```
X_train.shape : (4431, 128, 128, 3)
X_test.shape : (1108, 128, 128, 3)
y_train shape: (4431,)
a value from y_train: 8
```

Dupa ce avem datele in array-urile respective le vom converti in numpy array-uri pentru a fi compatibile cu functia de fit. De asemenea verificand forma, putem observa numarul de termeni din fiecare, dimensiunea imaginilor cat si cele 3 canale R,G,B la imagini.

2.2 Modelul 1

Pentru acest model am realizat pasii precizati anterior. Acesta este de tip secvential si are un singur layer de convolutie 2D cu activare 'relu' si MaxPooling2D cu un filtru de 2 pe 2.De asemenea acesta mai contine un layer de Flatten si unul Dense de dimensiune egala cu numarul claselor si activare 'softmax'

```
1
2  model = Sequential()
3
4  model.add(Conv2D(64, (3, 3), input_shape=X_train.shape[1:]))
5  model.add(Activation("relu"))
6  model.add(MaxPooling2D(pool_size=(2, 2)))
7
8  model.add(Flatten())
9
10 model.add(Dense(12))
11 model.add(Activation('softmax'))
12
13
14 print(model.summary())
```

Rolul unui layer Conv2D este de a crea un nucleu de convolutie, conform marimii specificate ca parametru, pentru a aplica o operatie de convolutie pe input. Cand acesta este folosit ca prim layer trebuie sa specificam si forma input-ului.

Rolul unui layer de activare este de a aplica o funtie de activare pe un output. In acest caz, functiile de activare sunt 'relu' si 'softmax'.
Functia 'relu' aplica max(x,0) pe toate elementele, adica daca avem elemente negative acestea vor deveni 0.
Functia 'softmax' converteste vectorul intr-o distributie de probabilitate si este de obicei folosita pe ultimul layer al clasificarilor pentru a interpreta rezultatul ca o distributie de probabilitati.

Rolul unul layer de MaxPooling2D este de a parcurge input-ul si cu un filtru de dimensiunea pool_size sa pastreze doar maximele din filtrul respectiv. Astfel, imaginea se va micsora si vor fi pastrate doar detaliile ce ies in evidenta.

Rolul unui layer de Flatten este de a modifica forma inputului, si de a o compresa termenii formei intr-un singur parametru. In cazul in care input-ul nu are suficienti parametri si este de forma (batch,), layer-ul de flatten va adauga un canal nou (batch,1).

Rolul unui layer Dense este de a conecta toti neuronii curenti, specificati in numarul dat ca parametru, cu toti neuronii din layer-ul anterior. Deci, este folosit pentru a schimba dimensiunea.

```
model.compile(loss="sparse_categorical_crossentropy", optimizer="adam", metrics=["accuracy"])
model_saved = model.fit(X_train, y_train_labels, epochs=5, validation_split=0.5)
loss, accuracy = model.evaluate(X_test, y_test_labels)
print("Loss: " + str(loss))
print("Accuracy: " + str(accuracy))
```

```
Epoch 1/5
70/70 [==============================] - 12s 172ms/step - loss: 231.6380 - accuracy: 0.1321 - val_loss: 2.0342 - val_accuracy: 0.3208
Epoch 2/5
70/70 [==============================] - 12s 167ms/step - loss: 1.6355 - accuracy: 0.4686 - val_loss: 1.8286 - val_accuracy: 0.4057
Epoch 3/5
70/70 [==============================] - 12s 172ms/step - loss: 1.0421 - accuracy: 0.6708 - val_loss: 1.9277 - val_accuracy: 0.4273
Epoch 4/5
70/70 [==============================] - 12s 172ms/step - loss: 0.6293 - accuracy: 0.8174 - val_loss: 2.1374 - val_accuracy: 0.4310
Epoch 5/5
70/70 [==============================] - 12s 171ms/step - loss: 0.4458 - accuracy: 0.8718 - val_loss: 2.5891 - val_accuracy: 0.4048
35/35 [==============================] - 1s 42ms/step - loss: 2.4106 - accuracy: 0.4070
Loss: 2.410551071166992
Accuracy: 0.40703970193862915
```

Dupa definirea modelului, acesta trebuie compilat si antrenat. La compilare definim functia de pierdere, optimizatorul si metrica.

In acest caz la functia de pierdere am folosit sparse_categorical_crossentropy, care calculeaza pierderea dintre label-uri si predictii. Se foloseste atunci cand avem 2 sau mai multe clase de label-uri.

Optimizer-ul implementeaza algoritmul Adam. Adam este o metoda SGD care se bazeaza pe estimarea adaptiva a momemtelor de ordinul 1 si 2. Este eficient, necesita memorie putina si e optim pentru problemele cu numar mare de parametrii.

Metrica calculeaza cat de des predictiile sunt egale cu label-urile. Aceasta creeaza doua variabile locale, total si count care sunt folosite pentru a calcula frecventa cu care y_pred e egal cu y_true.

Cand antrenam modelul trebuie sa specificam datele de train ca intrare, numarul de epoci care sa ruleze, dimensiunea batch-urilor si procentul din datele de training care sa fie asignate datelor pentru validare.

Dupa ce a fost antrenat calculam pierderea si acuratetea folosind datele de test definite la inceputul programului. Dupa cum se poate observa mai sus, valorile acestora nu sunt foarte optime.

2.3 Modelul 2

Modelul al doilea este asemanator cu primul. Acesta are in plus o operatie de normalizare pe toate imaginile inainte de a fi antrenat, operatie care aduce toate valorile in intervalul [0,1].

Pe langa aceasta, am mai adaugat un filtru de convolutie 2D cu activare 'relu' si MaxPooling2D cu filtrul de marime 3. Am mai adaugat si inca un layer dense de marime 36.

Singura diferenta la antrenare dintre acest model si modelul precedent este ca acesta are batch_size-ul specificat(daca nu este specificat va fi 1).

Se poate observa ca diferenta dintre rezultatele acestui model si a celui precedent nu sunt la amre distanta unul de celalalt.

```
In [28]:
model.compile(loss="sparse_categorical_crossentropy", optimizer="adam", metrics=["accuracy"])
model_saved = model.fit(X_train, y_train_labels, epochs=5, batch_size=16, validation_split=0.5)
loss, accuracy = model.evaluate(X_test, y_test_labels)
print("Loss: " + str(loss))
print("Accuracy: " + str(accuracy))
```

```
Epoch 1/5
139/139 [==============================] - 16s 113ms/step - loss: 1.4367 - accuracy: 0.5040 - val_loss: 1.5894 - val_accuracy: 0.4409
Epoch 2/5
139/139 [==============================] - 15s 108ms/step - loss: 1.4679 - accuracy: 0.4843 - val_loss: 1.5351 - val_accuracy: 0.4621
Epoch 3/5
139/139 [==============================] - 15s 109ms/step - loss: 1.3771 - accuracy: 0.5290 - val_loss: 1.6260 - val_accuracy: 0.4472
Epoch 4/5
139/139 [==============================] - 15s 110ms/step - loss: 1.3570 - accuracy: 0.5161 - val_loss: 1.5193 - val_accuracy: 0.4675
Epoch 5/5
139/139 [==============================] - 15s 110ms/step - loss: 1.3102 - accuracy: 0.5290 - val_loss: 1.4486 - val_accuracy: 0.4946
35/35 [==============================] - 2s 47ms/step - loss: 1.4093 - accuracy: 0.4937
Loss: 1.4093292951583862
Accuracy: 0.493682324886322
```

2.4 Modelul 3

La al treilea model, fata de modelul anterior, am mai adaugat inca un layer Conv2D, inca un layer Dense si am schimbat marimea nucleelor de max pooling la 2.

```
model.compile(loss="sparse_categorical_crossentropy", optimizer="adam", metrics=["accuracy"])
model_saved = model.fit(X_train, y_train_labels, epochs=5, batch_size=16, validation_split=0.5)
loss, accuracy = model.evaluate(X_test, y_test_labels)
print("Loss: " + str(loss))
print("Accuracy: " + str(accuracy))
```

```
Epoch 1/5
139/139 [==============================] - 45s 322ms/step - loss: 2.4287 - accuracy: 0.1524 - val_loss: 1.9504 - val_accuracy: 0.3533
Epoch 2/5
139/139 [==============================] - 45s 325ms/step - loss: 1.7638 - accuracy: 0.4225 - val_loss: 1.6598 - val_accuracy: 0.4328
Epoch 3/5
139/139 [==============================] - 44s 319ms/step - loss: 1.4973 - accuracy: 0.4818 - val_loss: 1.5742 - val_accuracy: 0.4589
Epoch 4/5
139/139 [==============================] - 44s 318ms/step - loss: 1.3265 - accuracy: 0.5359 - val_loss: 1.4542 - val_accuracy: 0.5113
Epoch 5/5
139/139 [==============================] - 45s 324ms/step - loss: 1.0272 - accuracy: 0.6334 - val_loss: 1.3305 - val_accuracy: 0.5745
35/35 [==============================] - 4s 116ms/step - loss: 1.3177 - accuracy: 0.5749
Loss: 1.3176687955856323
Accuracy: 0.5749097466468811
```

2.5 Modelul 4

La al patrulea model am schimbat toate functiile de activare in sigmoid, MaxPooling2D in AveragePooling2D si optimizer-ul in SGD. De asemenea am mai adaugat 3 layer-uri lambda. Functia de activare sigmoid va returna pentru valori mai mici decat -5 valoarea 0, respectiv pentru cele mai mari decat 5 valoarea 1 (sigmoid(x) = 1/(1+exp(-x))). AveragePooling2D ia valoarea medie din input pentru fiecare canal. Optimizer-ul SGD este un algoritm mult mai simplu decat Adam, acesta facand un update la parametrii pentru fiecare exemplu de training si label.

```
model.compile(loss="sparse_categorical_crossentropy", optimizer="sgd", metrics=["accuracy"])
model_saved = model.fit(X_train, y_train_labels, epochs=5, batch_size=16, validation_split=0.5)
loss, accuracy = model.evaluate(X_test, y_test_labels)
print("Loss: " + str(loss))
print("Accuracy: " + str(accuracy))
```

```
Epoch 1/5
139/139 [==============================] - 47s 337ms/step - loss: 2.6363 - accuracy: 0.0662 - val_loss: 2.4189 - val_accuracy: 0.1449
Epoch 2/5
139/139 [==============================] - 48s 345ms/step - loss: 2.4228 - accuracy: 0.1305 - val_loss: 2.4193 - val_accuracy: 0.1227
Epoch 3/5
139/139 [==============================] - 44s 315ms/step - loss: 2.4198 - accuracy: 0.1212 - val_loss: 2.4204 - val_accuracy: 0.1227
Epoch 4/5
139/139 [==============================] - 43s 313ms/step - loss: 2.4038 - accuracy: 0.1347 - val_loss: 2.4195 - val_accuracy: 0.1227
Epoch 5/5
139/139 [==============================] - 44s 319ms/step - loss: 2.4068 - accuracy: 0.1200 - val_loss: 2.4200 - val_accuracy: 0.1449
35/35 [==============================] - 5s 137ms/step - loss: 2.4163 - accuracy: 0.1282
Loss: 2.4162707328796387
Accuracy: 0.128158837556839
```

2.6 Modelul 5

Al 5-lea model este asemanator cu al 3-lea. Diferenta este la functiile de activare, am folosit in ordine elu (exponential linear unit), selu(scalar exponential linear unit),relu(rectified exponential linear unit), softplus, softsign si softmax. De asemenea ca optimizer am folosit Nadam(asemanator cu Adam dar acesta are si Gradient descent impuls Nesterov, o varianta imbunatatita de gradient descent)
relu(x): max(x,0)
elu(x): x daca x $>$ 0, alpha * (exp(x) - 1) daca x $<$ 0
selu(x): scale * elu(x, alpha)
softplus(x) = log(exp(x) + 1)
softsign(x) = x / (abs(x) + 1)

```
model.compile(loss="sparse_categorical_crossentropy", optimizer="nadam", metrics=["accuracy"])
model_saved = model.fit(X_train, y_train_labels, epochs=5, batch_size=16, validation_split=0.5)
loss, accuracy = model.evaluate(X_test, y_test_labels)
print("Loss: " + str(loss))
print("Accuracy: " + str(accuracy))
```

```
Epoch 1/5
139/139 [==============================] - 50s 352ms/step - loss: 2.2713 - accuracy: 0.2112 - val_loss: 1.6749 - val_accuracy: 0.4287
Epoch 2/5
139/139 [==============================] - 49s 353ms/step - loss: 1.4477 - accuracy: 0.5054 - val_loss: 1.3560 - val_accuracy: 0.5681
Epoch 3/5
139/139 [==============================] - 49s 351ms/step - loss: 1.0089 - accuracy: 0.6668 - val_loss: 1.6356 - val_accuracy: 0.4770
Epoch 4/5
139/139 [==============================] - 53s 379ms/step - loss: 0.7269 - accuracy: 0.7733 - val_loss: 1.1759 - val_accuracy: 0.6241
Epoch 5/5
139/139 [==============================] - 51s 370ms/step - loss: 0.4572 - accuracy: 0.8571 - val_loss: 1.1086 - val_accuracy: 0.6453
35/35 [==============================] - 4s 125ms/step - loss: 1.0851 - accuracy: 0.6462
Loss: 1.0850633382797241
Accuracy: 0.6462093591690063
```

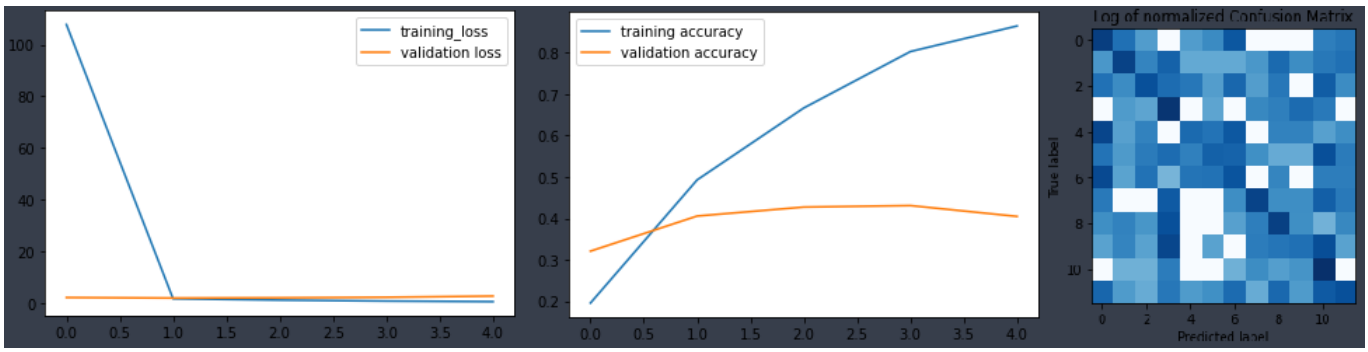
Chapter 3

Analiza rezultatelor

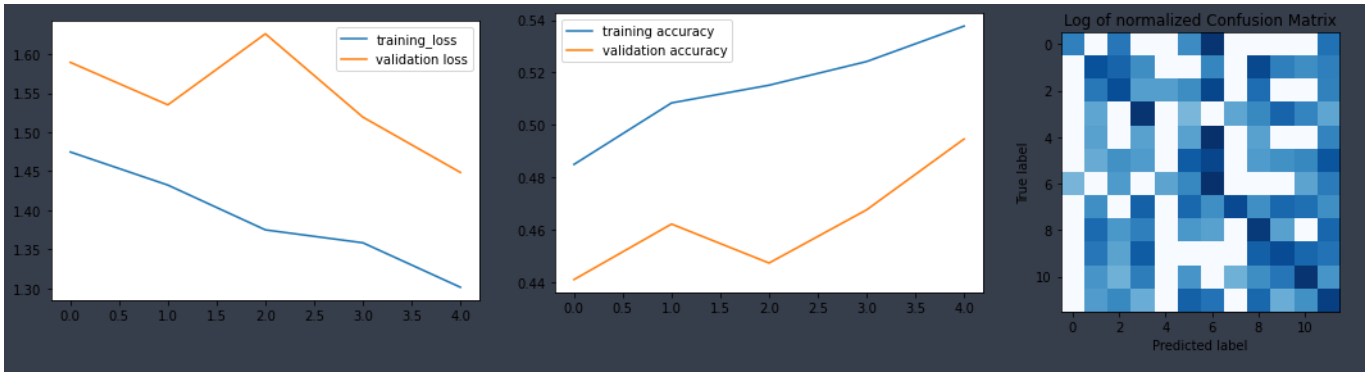
Mai jos am atasat graficele pentru evolutia pierderii si acuratetii modelelor de mai sus, cat si cate o matrice de confuzie pentru 10 elemente prezise, pentru fiecare model. Ordinea crescatoare a calitatii rezultatelor modelelor este urmatoarea: 4,1,2,3,5. Se poate observa la primele 3 modele faptul ca cresterea in acuratete este direct proportionala cu cresterea numarului de layere.

Pe de alta parte modelul 4 a fost conceput pentru a avea o acuratete cat mai mica si o pierdere cat mai mare. Rezultatele acestuia sunt datorita inlocuirii functiei de activare cu sigmoid, folosirea optimizatorului SGD in loc de Adam si adaugarea layer-elor lambda.

Nu in ultimul rand, modelul 5 are cele mai bune rezultate datorita schimbarii optimizatorului(Nadam) si schimbarea functiilor de activare(relu,elu,selu,softplus,softsign,softmax).

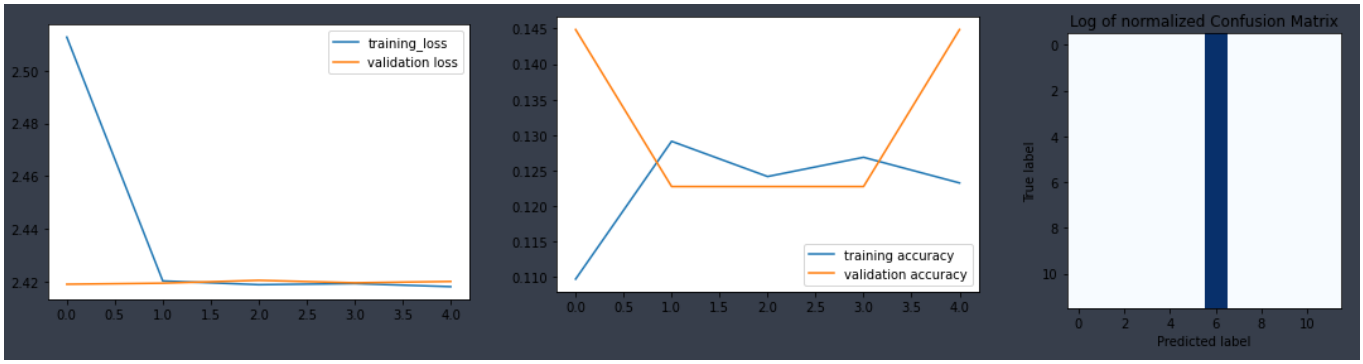


Rezultate model1

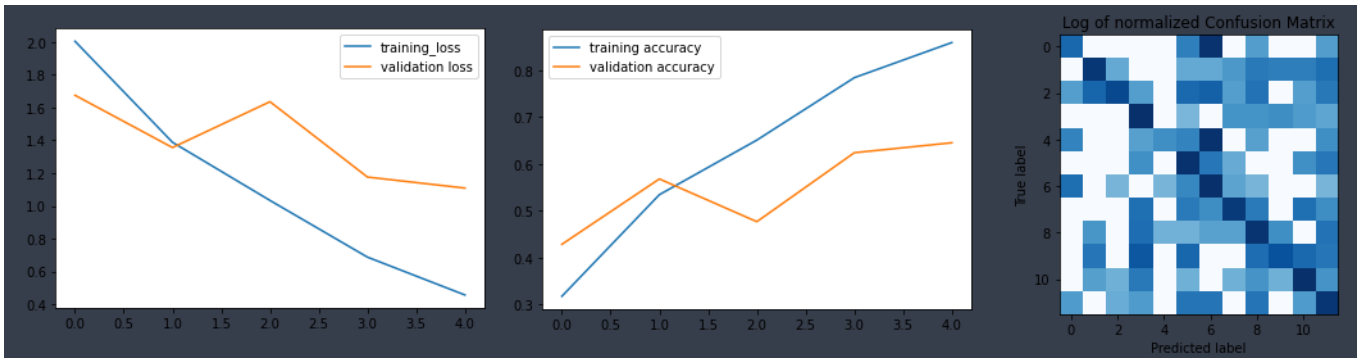


Rezultate model2

Rezultate model3



Rezultate model4



Rezultate model5

Chapter 4

Concluzii

In urma acestui proiect am implementat mai multe modele de retele neuronale convolutionale si am observat modul de functionare al acestora, cat si modul in care diferitele implementari afecteaza rezultatul si metricile de performanta. De asemenea am observat cat de mult este afectata valoarea rezultatului din cauza schimbarii unor parametrii.

Chapter 5

Anexa

```
1
 #!/usr/bin/env python
# coding: utf-8

# In[36]:


import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from sklearn.model_selection import train_test_split
import cv2 as cv
from tensorflow.keras.layers import Dense, Activation, Flatten, Conv2D,
    MaxPooling2D
import os
import matplotlib.pyplot as plt


# In[37]:


filenames = []
labels = []
classes = os.listdir('dataset')
for i in classes:
    imgs_name = os.listdir('E:/~SI/Plantling_Recognition2/dataset/' + i)
    for j in imgs_name:
        labels.append(int(i))
        filenames.append('E:/~SI/Plantling_Recognition2/dataset/' + i + '/'
    + j)

filenames = np.asarray(filenames)
labels = np.asarray(labels)
print(filenames)
print(labels)

print(filenames.shape)
print(labels.shape)

X_train_images, X_test_images, y_train_labels, y_test_labels =
    train_test_split(filenames,

        labels,

        test_size=0.2,
```

```
        random_state=45)


# In[38]:


print("Number of training images: ", len(X_train_images))
print("Number of training labels: ", len(y_train_labels))
print("Number of testing images: ", len(X_test_images))
print("Number of testing labels: ", len(y_test_labels))


# In[28]:


X_train = []
for i in range(len(X_train_images)):
    print(i)
    aux = cv.imread(X_train_images[i], cv.IMREAD_COLOR)
    aux = cv.resize(aux, (128, 128))
    X_train.append(aux)

X_test = []
for i in range(len(X_test_images)):
    print(i)
    aux = cv.imread(X_test_images[i], cv.IMREAD_COLOR)
    aux = cv.resize(aux, (128, 128))
    X_test.append(aux)


# In[29]:



X_train = np.asarray(X_train)
X_test = np.asarray(X_test)
print("X_train.shape : " + str(X_train.shape))
print("X_test.shape : " + str(X_test.shape))
# plt.imshow(X_train[0])
# plt.show()
print("y_train shape: " + str(y_train_labels.shape))
print("a value from y_train: " + str(y_train_labels[1]))


# In[30]:


# print("\nX_train[0] inainte de normalizare: " + str(X_train[0]) + "\n")
# X_train = tf.keras.utils.normalize(X_train, axis=1)
# X_test = tf.keras.utils.normalize(X_test, axis=1)
# print("\nX_train[0] dupa normalizare: " + str(X_train[0]) + "\n")


# In[31]:



model = Sequential()

```

```
model.add(Conv2D(64, (3, 3), input_shape=X_train.shape[1:]))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(5, 5)))

model.add(Flatten())

model.add(Dense(12))
model.add(Activation('softmax'))


print(model.summary())


# In[32]:



model.compile(loss="sparse_categorical_crossentropy", optimizer="adam",
    metrics=["accuracy"])
model_saved = model.fit(X_train, y_train_labels, epochs=5, validation_split
    =0.5)
loss, accuracy = model.evaluate(X_test, y_test_labels)
print("Loss: " + str(loss))
print("Accuracy: " + str(accuracy))


# In[33]:


plt.plot(model_saved.history['loss'] , label = 'training_loss')
plt.plot(model_saved.history['val_loss'], label = 'validation loss')
plt.legend()


# In[34]:


plt.plot(model_saved.history['accuracy'], label = 'training accuracy')
plt.plot(model_saved.history['val_accuracy'], label = 'validation accuracy')
plt.legend()


# In[35]:


predictions = model.predict(X_test, verbose=1)

pred_labels = []
for i in predictions:
    pred_labels.append(np.unique(labels)[np.argmax(i)])
pred_labels[:10]

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test_labels,pred_labels)
cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
cm = np.log(.0001 + cm)
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Log of normalized Confusion Matrix')
plt.ylabel('True label')
plt.xlabel('Predicted label')
```

```
plt.show()


# In[ ]:


```

Listing 5.1: model1

```


 #!/usr/bin/env python
# coding: utf-8

# In[21]:


import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from sklearn.model_selection import train_test_split
import cv2 as cv
from tensorflow.keras.layers import Dense, Activation, Flatten, Conv2D,
    MaxPooling2D
import os
import matplotlib.pyplot as plt


# In[22]:


filenames = []
labels = []
classes = os.listdir('dataset')
for i in classes:
    imgs_name = os.listdir('E:/~SI/Plantling_Recognition2/dataset/' + i)
    for j in imgs_name:
        labels.append(int(i))
        filenames.append('E:/~SI/Plantling_Recognition2/dataset/' + i + '/'
    + j)

filenames = np.asarray(filenames)
labels = np.asarray(labels)
print(filenames)
print(labels)

print(filenames.shape)
print(labels.shape)

X_train_images, X_test_images, y_train_labels, y_test_labels =
    train_test_split(filenames,

        labels,

        test_size=0.2,

        random_state=45)


# In[23]:


```

```
47
48 print("Number of training images: ", len(X_train_images))
49 print("Number of training labels: ", len(y_train_labels))
50 print("Number of testing images: ", len(X_test_images))
51 print("Number of testing labels: ", len(y_test_labels))
52
53
54 # In[24]:
55
56
57
58 X_train = []
59 for i in range(len(X_train_images)):
60     print(i)
61     aux = cv.imread(X_train_images[i], cv.IMREAD_COLOR)
62     aux = cv.resize(aux, (128, 128))
63     X_train.append(aux)
64
65 X_test = []
66 for i in range(len(X_test_images)):
67     print(i)
68     aux = cv.imread(X_test_images[i], cv.IMREAD_COLOR)
69     aux = cv.resize(aux, (128, 128))
70     X_test.append(aux)
71
72
73 # In[13]:
74
75
76
77 X_train = np.asarray(X_train)
78 X_test = np.asarray(X_test)
79 print("X_train.shape : " + str(X_train.shape))
80 print("X_test.shape : " + str(X_test.shape))
81 # plt.imshow(X_train[0])
82 # plt.show()
83 print("y_train shape: " + str(y_train_labels.shape))
84 print("a value from y_train: " + str(y_train_labels[1]))
85
86
87 # In[25]:
88
89
90 print("\nX_train[0] inainte de normalizare: " + str(X_train[0]) + "\n")
91 X_train = tf.keras.utils.normalize(X_train, axis=1)
92 X_test = tf.keras.utils.normalize(X_test, axis=1)
93 print("\nX_train[0] dupa normalizare: " + str(X_train[0]) + "\n")
94
95
96 # In[26]:
97
98
99
100 model = Sequential()
101
102 model.add(Conv2D(64, (3, 3), input_shape=X_train.shape[1:]))
103 model.add(Activation("relu"))
104 model.add(MaxPooling2D(pool_size=(5, 5)))
105
106 model.add(Conv2D(64, (3, 3)))
```

```
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(3, 3)))

model.add(Flatten())
model.add(Dense(36))
model.add(Activation("relu"))

model.add(Dense(12))
model.add(Activation('softmax'))


print(model.summary())


# In[28]:



model.compile(loss="sparse_categorical_crossentropy", optimizer="adam",
    metrics=["accuracy"])
model_saved = model.fit(X_train, y_train_labels, epochs=5, batch_size=16,
    validation_split=0.5)
loss, accuracy = model.evaluate(X_test, y_test_labels)
print("Loss: " + str(loss))
print("Accuracy: " + str(accuracy))


# In[29]:


plt.plot(model_saved.history['loss'] , label = 'training_loss')
plt.plot(model_saved.history['val_loss'], label = 'validation loss')
plt.legend()


# In[30]:


plt.plot(model_saved.history['accuracy'], label = 'training accuracy')
plt.plot(model_saved.history['val_accuracy'], label = 'validation accuracy')
plt.legend()


# In[31]:


predictions = model.predict(X_test, verbose=1)

pred_labels = []
for i in predictions:
    pred_labels.append(np.unique(labels)[np.argmax(i)])
pred_labels[:10]

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test_labels,pred_labels)
cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
cm = np.log(.0001 + cm)
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Log of normalized Confusion Matrix')
plt.ylabel('True label')
```

```
plt.xlabel('Predicted label')
plt.show()


# In[ ]:
```

Listing 5.2: model2

```


 #!/usr/bin/env python
# coding: utf-8

# In[1]:


import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from sklearn.model_selection import train_test_split
import cv2 as cv
from tensorflow.keras.layers import Dense, Activation, Flatten, Conv2D,
    MaxPooling2D
import os
import matplotlib.pyplot as plt


# In[2]:


filenames = []
labels = []
classes = os.listdir('dataset')
for i in classes:
    imgs_name = os.listdir('E:/~SI/Plantling_Recognition2/dataset/' + i)
    for j in imgs_name:
        labels.append(int(i))
        filenames.append('E:/~SI/Plantling_Recognition2/dataset/' + i + '/'
    + j)

filenames = np.asarray(filenames)
labels = np.asarray(labels)
print(filenames)
print(labels)

print(filenames.shape)
print(labels.shape)

X_train_images, X_test_images, y_train_labels, y_test_labels =
    train_test_split(filenames,

        labels,

```

```
        test_size=0.2,
42
        random_state=45)
43
44
45  # In[3]:
46
47
48  print("Number of training images: ", len(X_train_images))
49  print("Number of training labels: ", len(y_train_labels))
50  print("Number of testing images: ", len(X_test_images))
51  print("Number of testing labels: ", len(y_test_labels))
52
53
54  # In[4]:
55
56
57
58  X_train = []
59  for i in range(len(X_train_images)):
60      print(i)
61      aux = cv.imread(X_train_images[i], cv.IMREAD_COLOR)
62      aux = cv.resize(aux, (128, 128))
63      X_train.append(aux)
64
65  X_test = []
66  for i in range(len(X_test_images)):
67      print(i)
68      aux = cv.imread(X_test_images[i], cv.IMREAD_COLOR)
69      aux = cv.resize(aux, (128, 128))
70      X_test.append(aux)
71
72
73  # In[5]:
74
75
76
77  X_train = np.asarray(X_train)
78  X_test = np.asarray(X_test)
79  print("X_train.shape : " + str(X_train.shape))
80  print("X_test.shape : " + str(X_test.shape))
81  # plt.imshow(X_train[0])
82  # plt.show()
83  print("y_train shape: " + str(y_train_labels.shape))
84  print("a value from y_train: " + str(y_train_labels[1]))
85
86
87  # In[6]:
88
89
90  print("\nX_train[0] inainte de normalizare: " + str(X_train[0]) + "\n")
91  X_train = tf.keras.utils.normalize(X_train, axis=1)
92  X_test = tf.keras.utils.normalize(X_test, axis=1)
93  print("\nX_train[0] dupa normalizare: " + str(X_train[0]) + "\n")
94
95
96  # In[7]:
97
98
99
```

```
model = Sequential()

model.add(Conv2D(64, (3, 3), input_shape=X_train.shape[1:]))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(256))
model.add(Activation("relu"))

model.add(Dense(128))
model.add(Activation("relu"))

model.add(Dense(12))
model.add(Activation('softmax'))


print(model.summary())


# In[8]:



model.compile(loss="sparse_categorical_crossentropy", optimizer="adam",
    metrics=["accuracy"])
model_saved = model.fit(X_train, y_train_labels, epochs=5, batch_size=16,
    validation_split=0.5)
loss, accuracy = model.evaluate(X_test, y_test_labels)
print("Loss: " + str(loss))
print("Accuracy: " + str(accuracy))


# In[34]:


plt.plot(model_saved.history['loss'] , label = 'training_loss')
plt.plot(model_saved.history['val_loss'], label = 'validation loss')
plt.legend()


# In[35]:


plt.plot(model_saved.history['accuracy'], label = 'training accuracy')
plt.plot(model_saved.history['val_accuracy'], label = 'validation accuracy')
plt.legend()


# In[39]:


```

```
predictions = model.predict(X_test, verbose=1)

pred_labels = []
for i in predictions:
    pred_labels.append(np.unique(labels)[np.argmax(i)])
pred_labels[:10]

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test_labels,pred_labels)
cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
cm = np.log(.0001 + cm)
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Log of normalized Confusion Matrix')
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```

Listing 5.3: model3

```
 #!/usr/bin/env python
# coding: utf-8

# In[2]:


import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from sklearn.model_selection import train_test_split
import cv2 as cv
from tensorflow.keras.layers import Dense, Activation, Flatten, Conv2D,
    MaxPooling2D, Conv1D,AveragePooling2D
import os
import matplotlib.pyplot as plt


# In[3]:


filenames = []
labels = []
classes = os.listdir('dataset')
for i in classes:
    imgs_name = os.listdir('E:/~SI/Plantling_Recognition2/dataset/' + i)
    for j in imgs_name:
        labels.append(int(i))
        filenames.append('E:/~SI/Plantling_Recognition2/dataset/' + i + '/'
    + j)

filenames = np.asarray(filenames)
labels = np.asarray(labels)
print(filenames)
print(labels)

print(filenames.shape)
print(labels.shape)
```

```
36
37 X_train_images, X_test_images, y_train_labels, y_test_labels =
       train_test_split(filenames,
38
           labels,
39
           test_size=0.2,
40
           random_state=45)
41
42
43 # In[4]:
44
45
46 print("Number of training images: ", len(X_train_images))
47 print("Number of training labels: ", len(y_train_labels))
48 print("Number of testing images: ", len(X_test_images))
49 print("Number of testing labels: ", len(y_test_labels))
50
51
52 # In[5]:
53
54
55
56 X_train = []
57 for i in range(len(X_train_images)):
58     print(i)
59     aux = cv.imread(X_train_images[i], cv.IMREAD_COLOR)
60     aux = cv.resize(aux, (128, 128))
61     X_train.append(aux)
62
63 X_test = []
64 for i in range(len(X_test_images)):
65     print(i)
66     aux = cv.imread(X_test_images[i], cv.IMREAD_COLOR)
67     aux = cv.resize(aux, (128, 128))
68     X_test.append(aux)
69
70
71 # In[6]:
72
73
74
75 X_train = np.asarray(X_train)
76 X_test = np.asarray(X_test)
77 print("X_train.shape : " + str(X_train.shape))
78 print("X_test.shape : " + str(X_test.shape))
79 # plt.imshow(X_train[0])
80 # plt.show()
81 print("y_train shape: " + str(y_train_labels.shape))
82 print("a value from y_train: " + str(y_train_labels[1]))
83
84
85 # In[7]:
86
87
88
89 # IMG_SIZE = 128
90 # X_train = np.array(X_train).reshape(-1, IMG_SIZE, IMG_SIZE, 3)
91 # X_test = np.array(X_test).reshape(-1, IMG_SIZE, IMG_SIZE, 3)
```

```
# print("X_train.shape: " + str(X_train.shape))
# print("X_test.shape: " + str(X_test.shape))

print("\nX_train[0] inainte de normalizare: " + str(X_train[0]) + "\n")
X_train = tf.keras.utils.normalize(X_train, axis=1)
X_test = tf.keras.utils.normalize(X_test, axis=1)
print("\nX_train[0] dupa normalizare: " + str(X_train[0]) + "\n")


# In[8]:



model = Sequential()

model.add(Conv2D(64, (3, 3), input_shape=X_train.shape[1:]))
model.add(Activation("sigmoid"))
model.add(AveragePooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation("sigmoid"))
model.add(AveragePooling2D(pool_size=(2, 2)))

model.add(tf.keras.layers.Lambda(lambda x: x ** 4))
model.add(tf.keras.layers.Lambda(lambda x: x ** 3))
model.add(tf.keras.layers.Lambda(lambda x: x ** 2))

model.add(Flatten())
model.add(Dense(128))
model.add(Activation("sigmoid"))

model.add(Dense(64))
model.add(Activation("sigmoid"))

model.add(Dense(12))
model.add(Activation('sigmoid'))


print(model.summary())


# In[9]:



model.compile(loss="sparse_categorical_crossentropy", optimizer="sgd",
    metrics=["accuracy"])
model_saved = model.fit(X_train, y_train_labels, epochs=5, batch_size=16,
    validation_split=0.5)
loss, accuracy = model.evaluate(X_test, y_test_labels)
print("Loss: " + str(loss))
print("Accuracy: " + str(accuracy))


# In[10]:


plt.plot(model_saved.history['loss'] , label = 'training_loss')
plt.plot(model_saved.history['val_loss'], label = 'validation loss')
plt.legend()
```

```

# In[11]:


plt.plot(model_saved.history['accuracy'], label = 'training accuracy')
plt.plot(model_saved.history['val_accuracy'], label = 'validation accuracy')
plt.legend()


# In[12]:


predictions = model.predict(X_test, verbose=1)

pred_labels = []
for i in predictions:
    pred_labels.append(np.unique(labels)[np.argmax(i)])
pred_labels[:10]

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test_labels,pred_labels)
cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
cm = np.log(.0001 + cm)
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Log of normalized Confusion Matrix')
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()


# In[ ]:




```

Listing 5.4: model4

```

  #!/usr/bin/env python
# coding: utf-8

# In[1]:


import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from sklearn.model_selection import train_test_split
import cv2 as cv
from tensorflow.keras.layers import Dense, Activation, Flatten, Conv2D,
    MaxPooling2D
import os
import matplotlib.pyplot as plt


# In[2]:


filenames = []
```

```
labels = []
classes = os.listdir('dataset')
for i in classes:
    imgs_name = os.listdir('E:/~SI/Plantling_Recognition2/dataset/' + i)
    for j in imgs_name:
        labels.append(int(i))
        filenames.append('E:/~SI/Plantling_Recognition2/dataset/' + i + '/'
   + j)

filenames = np.asarray(filenames)
labels = np.asarray(labels)
print(filenames)
print(labels)

print(filenames.shape)
print(labels.shape)

X_train_images, X_test_images, y_train_labels, y_test_labels =
   train_test_split(filenames,

        labels,

        test_size=0.2,

        random_state=45)


# In[3]:


print("Number of training images: ", len(X_train_images))
print("Number of training labels: ", len(y_train_labels))
print("Number of testing images: ", len(X_test_images))
print("Number of testing labels: ", len(y_test_labels))


# In[4]:


X_train = []
for i in range(len(X_train_images)):
    print(i)
    aux = cv.imread(X_train_images[i], cv.IMREAD_COLOR)
    aux = cv.resize(aux, (128, 128))
    X_train.append(aux)

X_test = []
for i in range(len(X_test_images)):
    print(i)
    aux = cv.imread(X_test_images[i], cv.IMREAD_COLOR)
    aux = cv.resize(aux, (128, 128))
    X_test.append(aux)


# In[5]:


X_train = np.asarray(X_train)
X_test = np.asarray(X_test)
print("X_train.shape : " + str(X_train.shape))
```

```
77 print("X_test.shape : " + str(X_test.shape))
78 # plt.imshow(X_train[0])
79 # plt.show()
80 print("y_train shape: " + str(y_train_labels.shape))
81 print("a value from y_train: " + str(y_train_labels[1]))
82
83
84 # In[6]:
85
86
87 print("\nX_train[0] inainte de normalizare: " + str(X_train[0]) + "\n")
88 X_train = tf.keras.utils.normalize(X_train, axis=1)
89 X_test = tf.keras.utils.normalize(X_test, axis=1)
90 print("\nX_train[0] dupa normalizare: " + str(X_train[0]) + "\n")
91
92
93 # In[7]:
94
95
96
97 model = Sequential()
98
99 model.add(Conv2D(64, (3, 3), input_shape=X_train.shape[1:]))
100 model.add(Activation("elu"))
101 model.add(MaxPooling2D(pool_size=(2, 2)))
102
103 model.add(Conv2D(64, (3, 3)))
104 model.add(Activation("selu"))
105 model.add(MaxPooling2D(pool_size=(2, 2)))
106
107 model.add(Conv2D(64, (3, 3)))
108 model.add(Activation("relu"))
109 model.add(MaxPooling2D(pool_size=(2, 2)))
110
111 model.add(Flatten())
112 model.add(Dense(128))
113 model.add(Activation("softplus"))
114
115 model.add(Dense(64))
116 model.add(Activation("softsign"))
117
118 model.add(Dense(12))
119 model.add(Activation('softmax'))
120
121
122 print(model.summary())
123
124
125 # In[8]:
126
127
128
129 model.compile(loss="sparse_categorical_crossentropy", optimizer="nadam",
      metrics=["accuracy"])
130 model_saved = model.fit(X_train, y_train_labels, epochs=5, batch_size=16,
      validation_split=0.5)
131 loss, accuracy = model.evaluate(X_test, y_test_labels)
132 print("Loss: " + str(loss))
133 print("Accuracy: " + str(accuracy))
134
```

```
135
136 # In[9]:
137
138
139 plt.plot(model_saved.history['loss'] , label = 'training_loss')
140 plt.plot(model_saved.history['val_loss'], label = 'validation loss')
141 plt.legend()
142
143
144 # In[10]:
145
146
147 plt.plot(model_saved.history['accuracy'], label = 'training accuracy')
148 plt.plot(model_saved.history['val_accuracy'], label = 'validation accuracy')
149 plt.legend()
150
151
152 # In[11]:
153
154
155 predictions = model.predict(X_test, verbose=1)
156
157 pred_labels = []
158 for i in predictions:
159     pred_labels.append(np.unique(labels)[np.argmax(i)])
160 pred_labels[:10]
161
162 from sklearn.metrics import confusion_matrix
163 cm = confusion_matrix(y_test_labels,pred_labels)
164 cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
165 cm = np.log(.0001 + cm)
166 plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
167 plt.title('Log of normalized Confusion Matrix')
168 plt.ylabel('True label')
169 plt.xlabel('Predicted label')
170 plt.show()
171
172
173
```

Listing 5.5: model5