Documentatie CT-scan classification

Stan Bianca-Mihaela

1 Cerinta

Vom clasifica un set de date in 3 categorii: "native", "arterial" si "venous". Aceasta problema de clasificare se numeste multi-class image classification.

2 Abordari

2.1 k-nearest neighbors algorithm (KNN)

Prima abordare a fost un clasificator KNN, inspirat din cel prezentat in laboratorul 3, cu 5 vecini.

Acesta obtine o acuratete pe datele de validare de 42,6% iar pe datele de test 39.794%.

Matricea de confuzie pentru aceasta abordare este:

```
[[632 547 321]
 [375 677 448]
 [291 601 608]]
```

2.2 Retele neuronale convolutionale (CNN)

2.2.1 Preprocesare

Pentru preprocesare am comparat performanta modelului cu date normalizate fata de performanta cu date standardizate.

Normalizarea implica transpunerea valorilor pixelilor din intervalul [0,255] in intervalul [0,1].

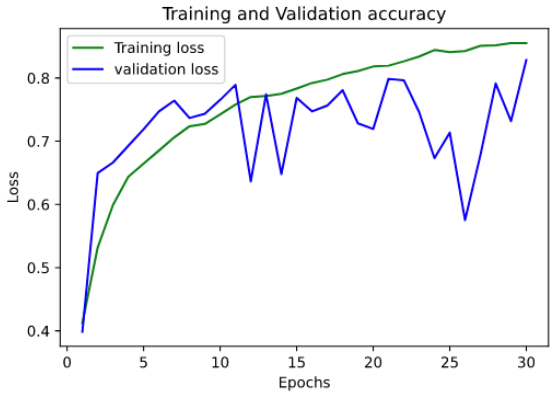
Standardizarea implica procesarea datelor astfel incat media sa fie 0 iar deviatia standard este 1. Formula pentru standardizare este:

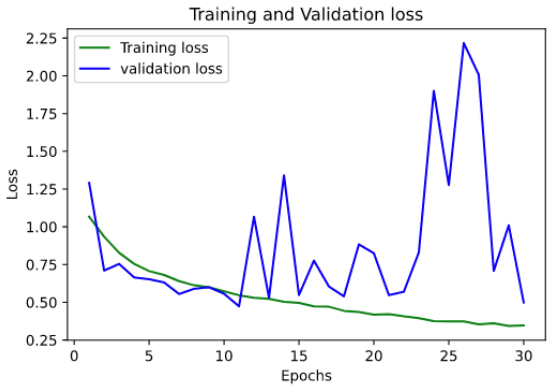
$$X' = \frac{X - \mu}{\sigma}$$

Pentru a obtine acesta standardizare voi folosi parametrii `featurewise_center` si `featurewise_std_normalization` din `ImageDataGenerator`

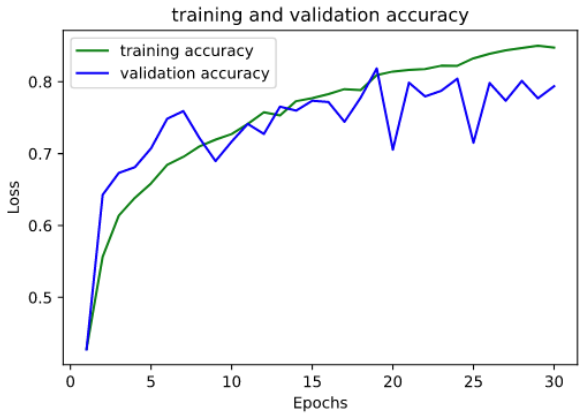
Voi compara efectele standardizarii datelor asupra performantei pentru modelul ales.

- fara standardizare, doar cu normalizarea datelor:





- cu standardizare:



Cum modelul antrenat pe datele standardizate pare a avea o invatare mai "smooth", voi standardiza datele.

2.2.2 Augumentarea datelor

Am realizat augumentarea datelor pe datele de antrenare prin intermediul unui `ImageDataGenerator` din `keras.preprocessing.image`. Metodele folosite sunt:

- Rotire cu pana la 30 de grade in orice directie.
- Shiftare catre dreapta sau stanga cu pana la 5 pixeli.
- Shiftare in sus sau in jos cu pana la 5 pixeli.
- Distorsionarea imaginilor cu pana la 10 grade.
- Intoarcerea random a imaginilor pe axa orizontala.
- Adaugarea unui layer care variaza contrastul in model.

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range = 30,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=10,
    zoom_range=0.1,
```

```
    horizontal_flip=True,
    featurewise_center=True,
    featurewise_std_normalization=True,
)

validation_datagen = ImageDataGenerator(
    rescale=1./255,
    featurewise_center=True,
    featurewise_std_normalization=True,
)
```

2.2.3 2D convolution layers

O convolutie aplica un filtru unui input astfel incat obine un map de feature-uri. Filtrul este aplicat printr-o matrice numita kernel care este glisata de-a lungul imaginii.

Pentru convolutii am folosit `Conv2D` din `keras.layers`.

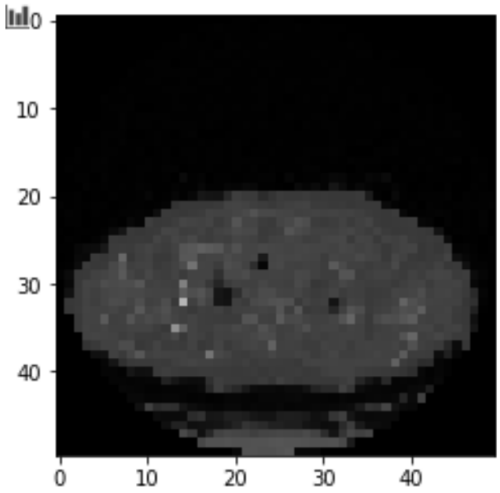


Figure 1: Imaginea originala.

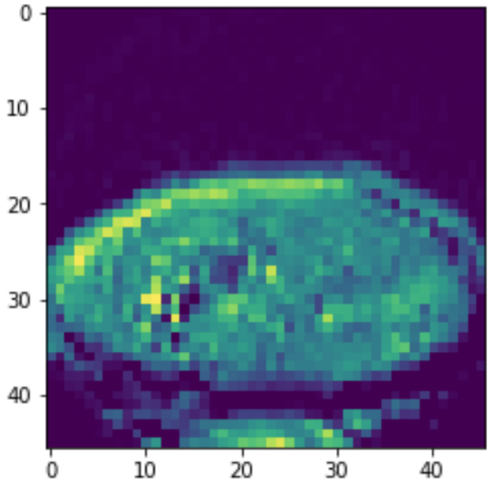


Figure 2: Imaginea dupa un layer convolutional.

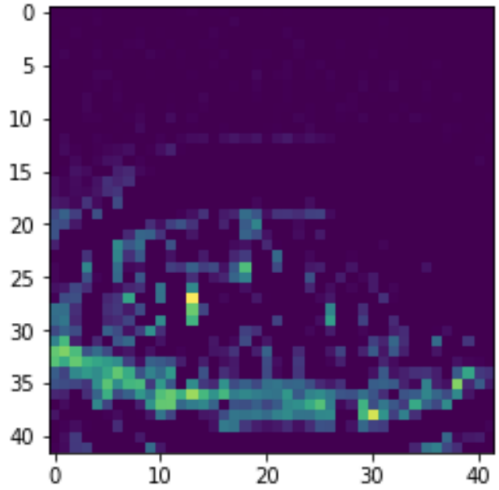


Figure 3: Imaginea dupa 2 layere de convolutie.

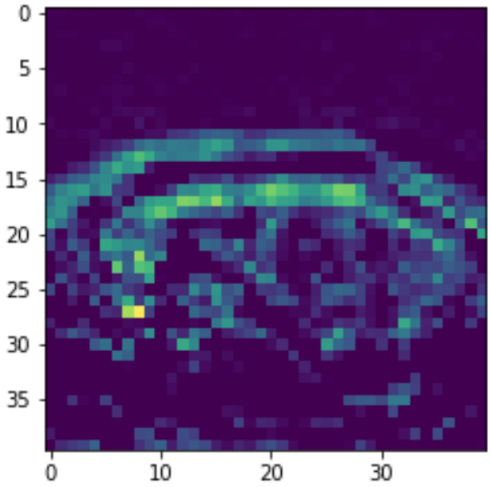
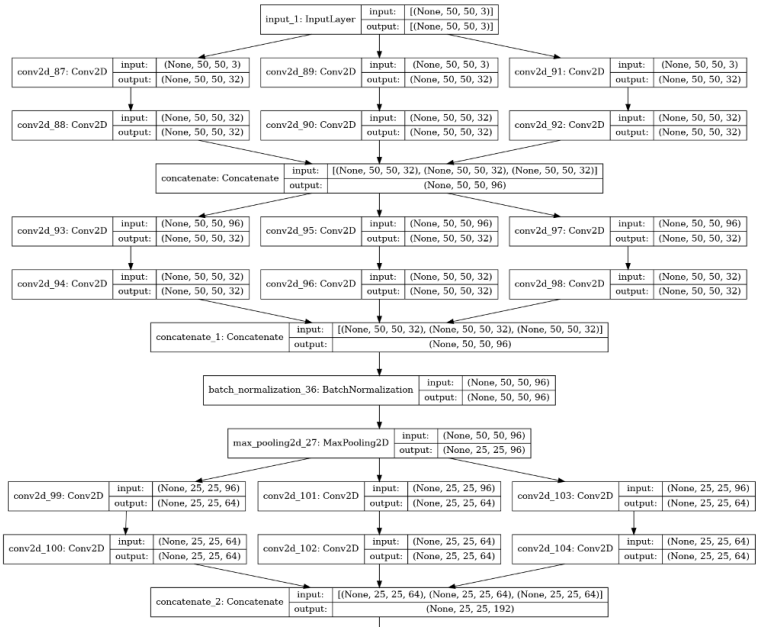


Figure 4: Imaginea dupa 3 layere de convolutie.

2.2.4 Alegerea unui model

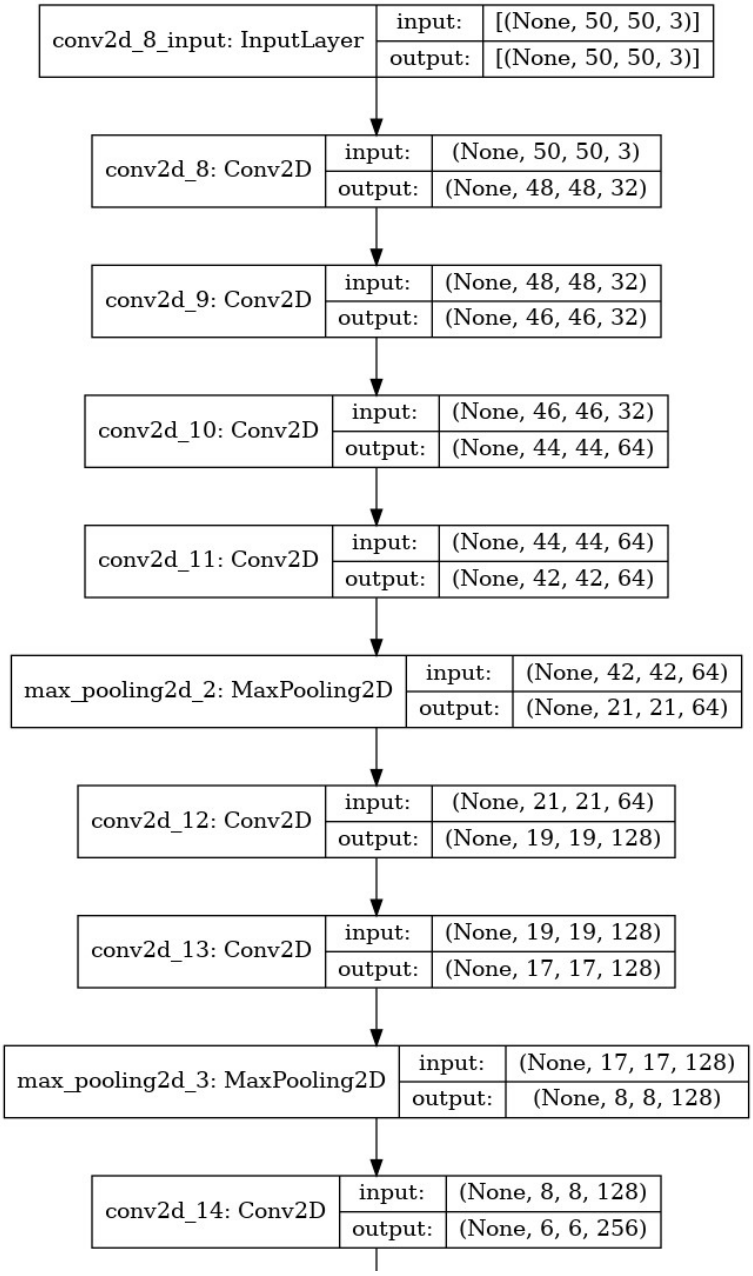
Am implementat diferite modele, printre care:

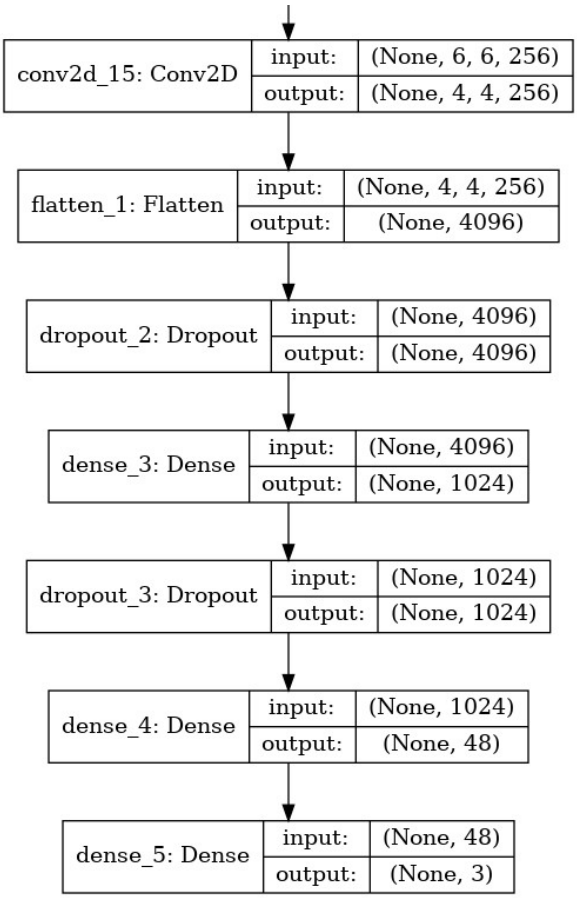
- Cateva modele bazate pe transfer learning, folosind modele pre-antrentate precum AlexNet, ResNet50 si EfficientNetB0 care au obtinut o acuratete de 50% -60%.
- O implementare a ResNet18 aplicata pe o varianta a dataset-ului cu rezolutia imbunatatita. Imbunatatirea s-a realizat prin super-resolution folosind modelul RRDN (Residual in Residual Dense Network) din libraria `cv2`. Imaginile au fost imbunatatite la o rezolutie de (200,200). Am ales reteaua ResNet18 pentru ca avea cele mai putine layere si era cea mai usor computational de antrenat de la zero. Aceasta abordare a fost inspirata de studiul [1] si a obtinut o acuratete de peste 70%, insa am renuntat la aceasta idee deoarece necesita o putere prea mare de procesare.
- Cateva modele bazate pe ramificarea modelului si concatenerea ulterioara a acelor ramuri pentru a sustrage diferite tipuri de features.



  Acest model si variatii ale lui ajungeu la performante de peste 70%, insa aveu o complexitate computationala mai mare iar rezultatele erau comparabile si chiar mai bune in cazul modelului final ales.

Modelul final ales are cele mai bune rezultate dintre cele abordate, ajungand in in mod consistent la o acuratete de peste 80%.

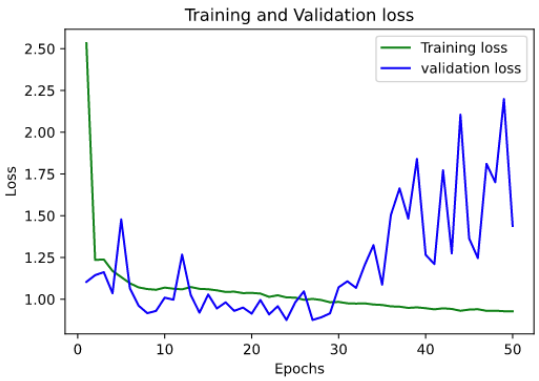

conv2d_8_input: InputLayer
input: [(None, 50, 50, 3)]
output: [(None, 50, 50, 3)]
conv2d_8: Conv2D
input: (None, 50, 50, 3)
output: (None, 48, 48, 32)
conv2d_9: Conv2D
input: (None, 48, 48, 32)
output: (None, 46, 46, 32)
conv2d_10: Conv2D
input: (None, 46, 46, 32)
output: (None, 44, 44, 64)
conv2d_11: Conv2D
input: (None, 44, 44, 64)
output: (None, 42, 42, 64)
max_pooling2d_2: MaxPooling2D
input: (None, 42, 42, 64)
output: (None, 21, 21, 64)
conv2d_12: Conv2D
input: (None, 21, 21, 64)
output: (None, 19, 19, 128)
conv2d_13: Conv2D
input: (None, 19, 19, 128)
output: (None, 17, 17, 128)
max_pooling2d_3: MaxPooling2D
input: (None, 17, 17, 128)
output: (None, 8, 8, 128)
conv2d_14: Conv2D
input: (None, 8, 8, 128)
output: (None, 6, 6, 256)

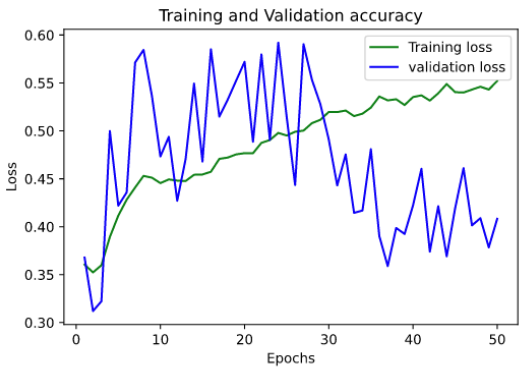


2.2.5 Functii de activare folosite

O functie de activare este o functie care prelucreaza suma ponderilor care intra intr-un nod.

Graficele modelului ales de mine fara a folosi o activare (exceptamd softmax-ul din ultimul layer) sunt:





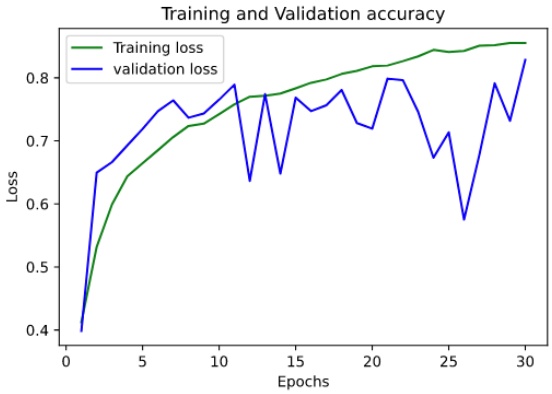
Softmax Functia de softmax imparte exponentierea output-urilor fiecarui neuron la suma probabilitatilor pentru toate clasele. Astfel, suma output-urilor din softmax este 1.

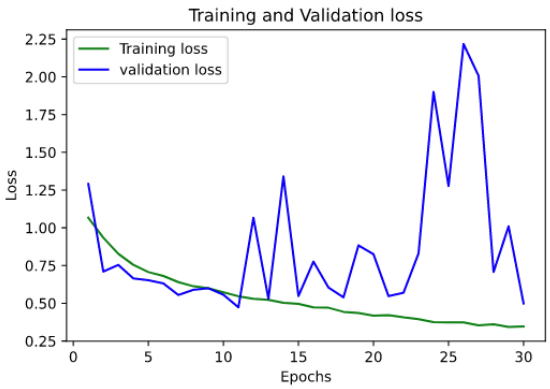
$$f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}}$$

Relu Functia de activare Relu (Rectified Linear Unit) este o functie monotona care "aduce la 0" valorile negative.

$$f(x) = \begin{cases} 0 & x \leq 0 \\ x & x > 0 \end{cases}$$

In urma adaugarii activatii Relu pe toate layerele, graficele sunt:





2.2.6 Categorical cross-entropy loss

Functia de pierdere categorical cross-entropy loss este of functie de pierdere care se foloseste pentru clasificarea in clase multiple. Aceasta functie are doi pasi:

- Aplicarea functiei de softmax.
- Aplicarea functiei de cross-entropy loss.

Formula pentru catecogical cross-entropy loss este:

$$CE = -\sum_{i}^{C} t_i log(f(s)_i)$$

unde:

- C este numarul de clase in care clasificam
- t_i este 1 doar daca clasa i este cea corecta.
- f este functia de softmax
- s este output-ul fiecarui neuron

Deoarece am transformat label-urile in one-hot encoding, am folosit categorical cross-entropy loss. Daca utilizam label-urile asa cum au fost date in dataset, as fi folosit sperse categorical cross-entropy loss.

2.2.7 Optimizatorul

Algoritmii de optimizare sunt metode de a gasi minimul unei functii. In cazul nostru, functia ce trebuie minimizata este loss-ul pe training data.

Gradient descent si stochastic gradient descent Gradient descent sa misca iterativ in directa celei mai abrupte imbunatatiri ale functiei de loss cu un pas egal cu learning rate. Stochastic gradient descent nu foloseste tot dataset-ul de train pentru a alege directa, ci foloseste un sample sau o submultime de samples.

Acceleratia Un optimizator poate incorpora o acceleratie. Aceasta adauga un parametru in plus ecuatiei care modifica parametri, si anume timpul. Pentru urmatoarea functie de stochastic gradient descent:

$$\theta = \theta - \alpha \nabla f(\theta)$$

unde:

- θ este parametrul (ponderi, biases, activari)
- α este learning rate-ul
- f este functia care trebuie minimizata

adaugarea acceleratiei arata astfel:

$$\theta = \theta - \alpha \nabla f(\theta) + \gamma \delta\theta$$

unde:

- γ este constanta acceleratie
- $\Delta\theta$ este ultima modificare a lui θ

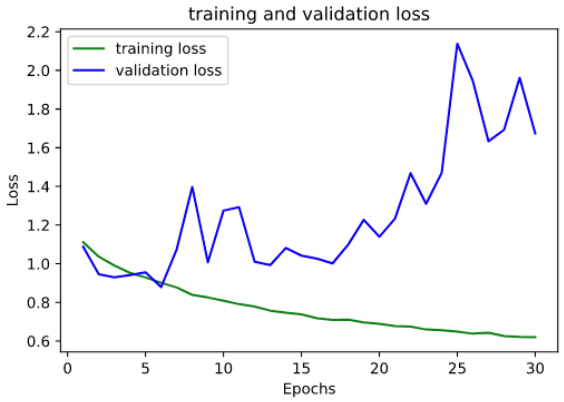
Adagrad Optimizatorul Adagrad este un optimizator care foloseste un learning rate adaptiv.
Astfel, dearning rate-ul devine acum:

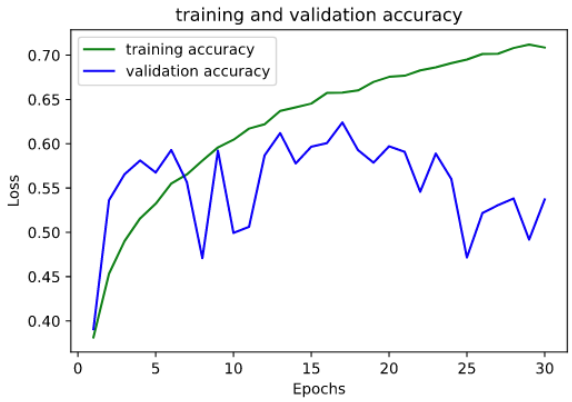
$$\alpha'_t = \frac{\alpha}{\sqrt{\alpha_t + \epsilon}}$$

unde:

- ϵ este un numar mic si pozitiv pentru a se evita impartirea la 0
- α_t este suma gradientilor ultimilor t gradienti la patrat

Graficele modelului cu optimizatorul Adagrad sunt:





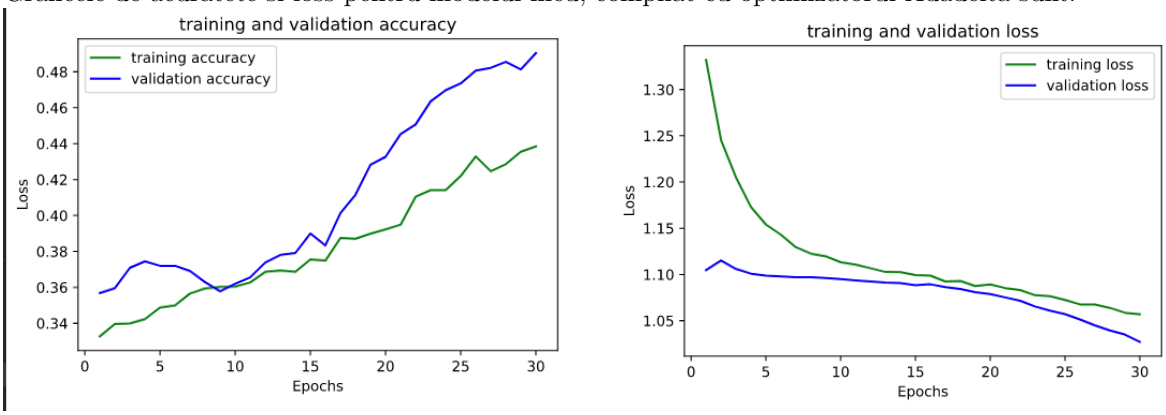
Adadelta Problama cu Adagrad este ca learning rate-ul se micoreaza pe masura ce se apropie de minim. Adadelta incearca sa rezolve aceasta problema prin restrange intervalul pentru care calculam suma gradientilor la patrat. Astfel:

$$\alpha'_t = \frac{\alpha}{S_{dw_t} + \epsilon}$$

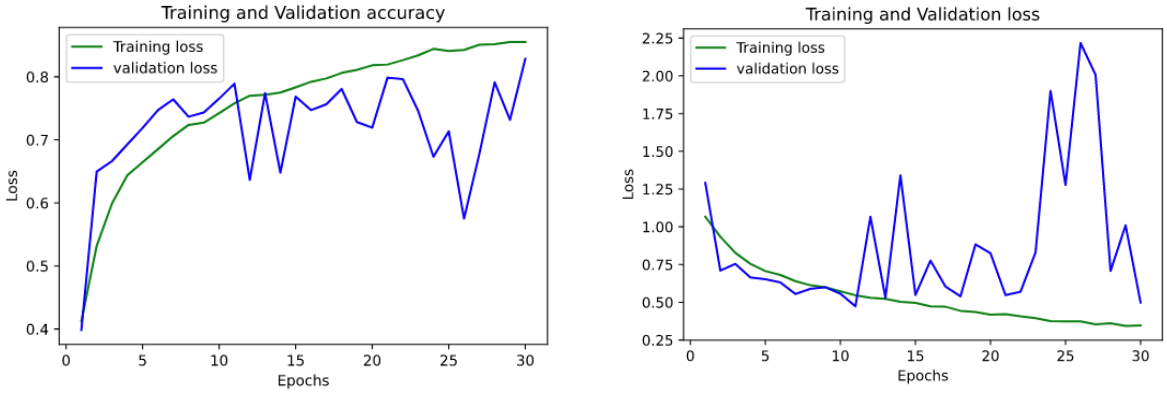
unde:

- $S_{dw_t} = \beta S_{dw_{t-1}} + (1 - \beta)x$, unde x este suma gradientilor la patrat.

Graficele de acuratete si loss pentru modelul meu, compilat cu optimizatorul Adadelta sunt:



Adam optimizatorul adam combina ideea de acceleratie cu adaptarea learning rate-ului din Adadelta.
Graficele de acuratete si loss pentru modelul meu, compilat cu optimizatorul Adam sunt:



Alegerea unui optimizator Considerand toate graficele studiate folosin cei 3 optimizatori am ales sa folosesc optimizatorul Adam pentru ca acesta atunge cea mai mare acuratete in cel mai scurt timp (30 epoci).

2.2.8 Tehnici de evitare a overfiting-ului

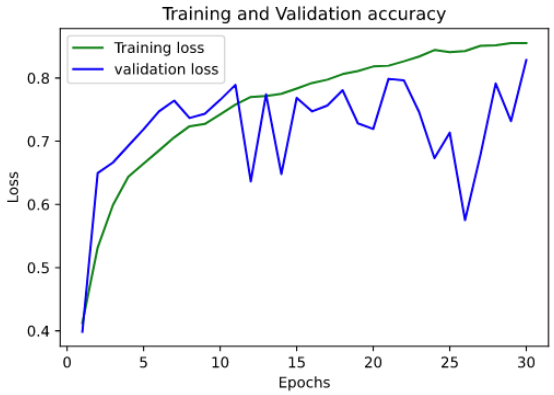
Metodele utilizate de mine pentru reducerea overfitting-ului sunt:

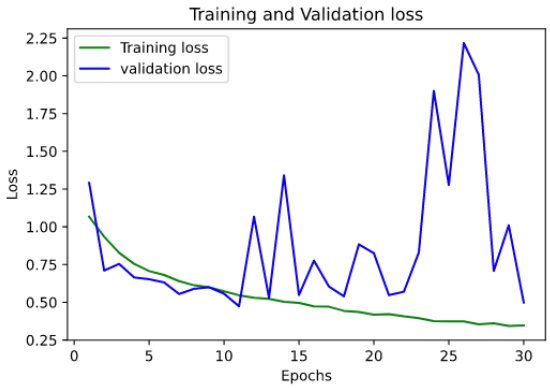
- Folosirea augumentarii pe date.
- Folosirea normalizarii pe batch-uri.
- Folosirea unui dropout pe layer-ele dense.

2.2.9 Dimensiunea unui batch

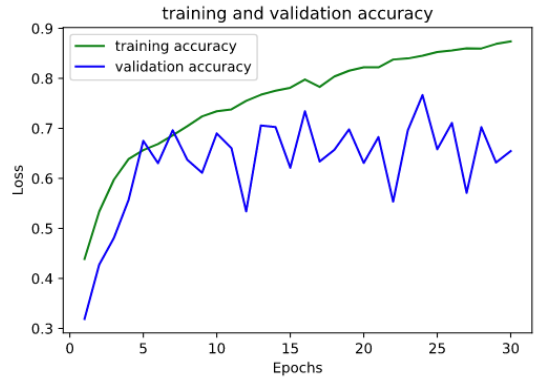
Voi varia dimensiunea unui batch pentru a optimiza performanta modelului:

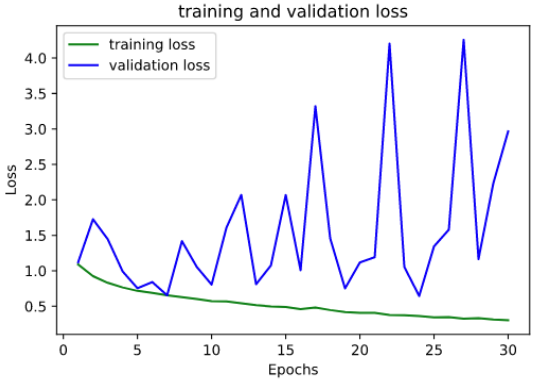
- batch = 32:





- batch = 96:





Asadar, am ales un batch size de 32.

2.2.10 Batch normalization

Batch normalization normalizeaza datele din layer-ul pe care este aplicat, relativ la batch-ul curent. Aceasta permite o invatare mai rapida si ofera un anumit grad de regularizare modelului datorita randomizarii batch-ului.

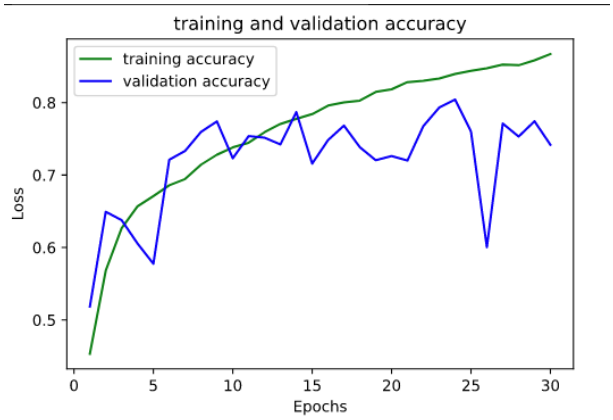
Fara batch normalization reteaua aleasa de mine invata foarte incet.

Pentru batch normalization am folosit `BatchNormalization` din `keras.layers`.

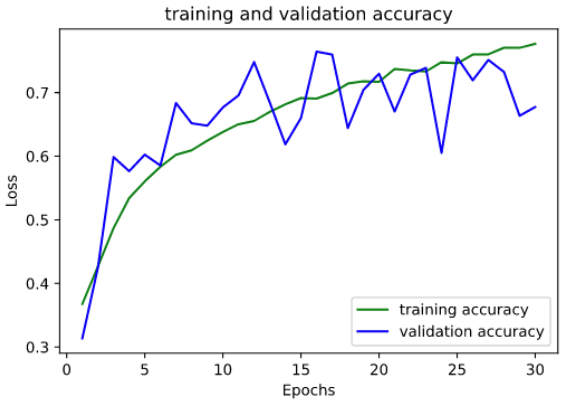
2.2.11 Dropout layers

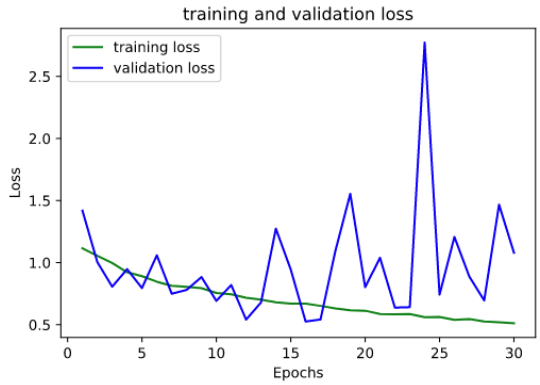
Layer-ele de dropout sunt o metoda de evitare a overfitting-ului prin dezactivarea anumitor neuroni din retea.

- Fara dropout:



- Cu dropout:





Pentru dropout am folosit `Dropout` din `keras.layers`.

2.2.12 Checkpoint-uri

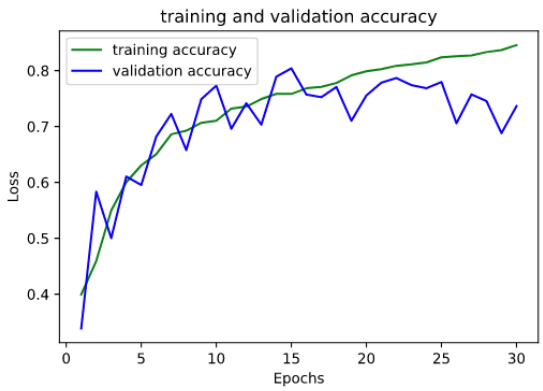
Pentru a salva ponderile modelului atunci cand acuratetea pe datele de validare este maxima, am folosit `ModelCheckpoint` din `keras.callbacks`.

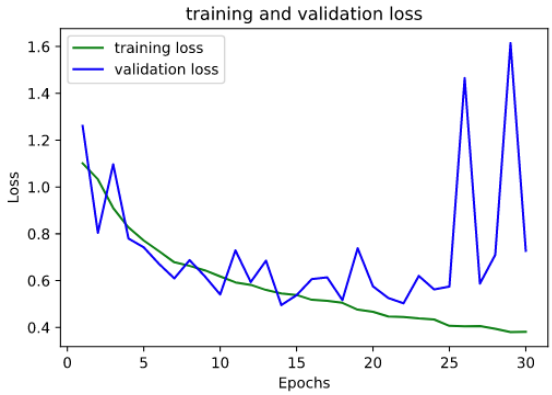
```
checkpoint_filepath = "checkpoint1"
model_checkpoint_callback1 = keras.callbacks.ModelCheckpoint(
    filepath = checkpoint_filepath,
    save_weights_only = True,
    monitor = 'val_acc',
    mode = 'max',
    save_best_only = True
)
```

2.2.13 Alegerea hiperparametrilor

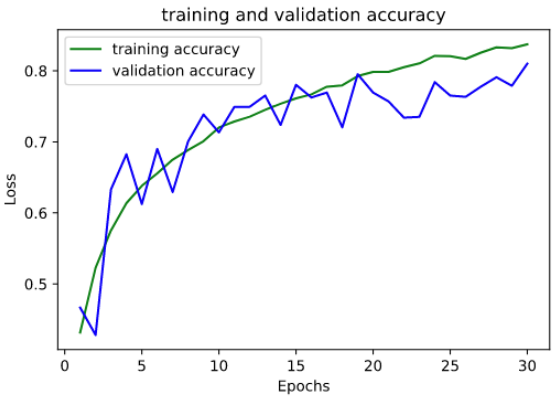
Am facut o serie de modificari modelului de baza si am testat performanta:

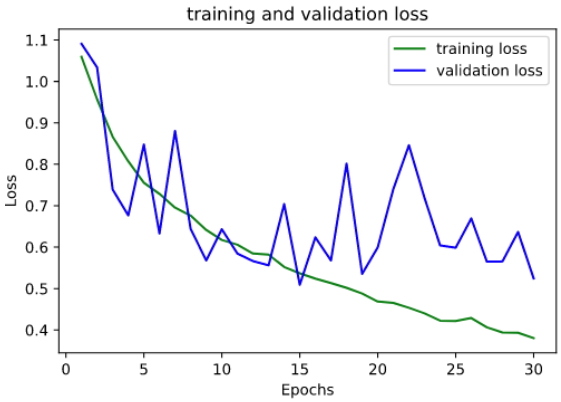
- Modificarea primului layer convolutional din `model.add(Conv2D(32,3, activation="relu", input_shape=(50, 50, 3,)))` in `model1.add(Conv2D(32,5, activation="relu", input_shape=(50, 50, 3,)))` si am obtinut o acuratete maxima de 80.40%.



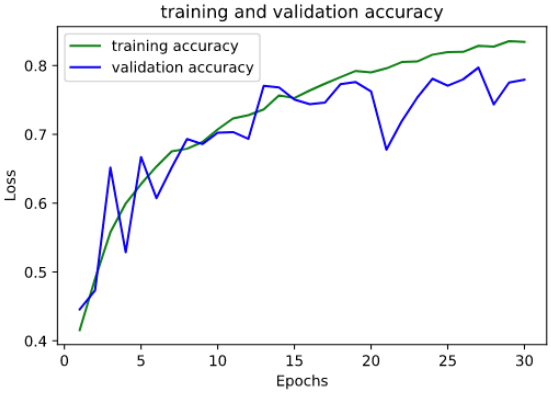


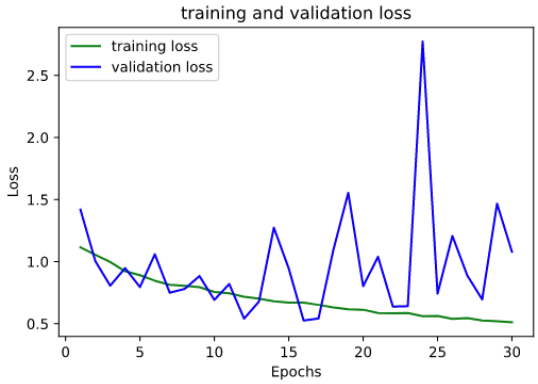
- Am modificat si al doilea layer al modelului din `model.add(Conv2D(32,3, activation="relu"))` in `model.add(Conv2D(32,5, activation="relu"))` si am obtinut o acuratete de de 81%.





- Modificarea primului layer convolutional din `model.add(Conv2D(32,5, activation="relu", input_shape=(50, 50, 3,)))` in `model1.add(Conv2D(32,7, activation="relu", input_shape=(50, 50, 3,)))` si am obtinut o acuratete maxima de 79.71%.





2.2.14 CNN-ul final

CNN-ul final este:

```
model = Sequential()
model.add(keras.layers.experimental.preprocessing.
RandomContrast(factor = 0.4, input_shape=(50, 50, 3,)))
model.add(Conv2D(32,5, activation="relu", input_shape=(50, 50, 3,)))
```

```
model.add(Conv2D(32,5,activation="relu"))
model.add(BatchNormalization())
model.add(Conv2D(64,3, activation="relu"))
model.add(Conv2D(64,3,activation="relu"))
model.add(BatchNormalization())
model.add(MaxPooling2D())
model.add(Conv2D(128,3,activation="relu"))
model.add(Conv2D(128,3,activation="relu"))
model.add(BatchNormalization())
model.add(MaxPooling2D())
model.add(Conv2D(256,3,activation="relu"))
model.add(Conv2D(256,3,activation="relu"))
model.add(Flatten())
model.add(Dropout(0.5))
model.add(Dense(1024, activation="relu"))
model.add(Dropout(0.5))
model.add(Dense(48, activation="relu"))
model.add(Dense(3, activation="softmax"))
model.summary()
model.compile(optimizer="adam", loss='categorical_crossentropy', metrics=['acc'])
```

Acesta a avut o acuratete de 85.04% pe datele de validare cu un loss de 0.4849 dupa aproximativ 80 de epoci, iar acuratetea in competitie a fost de 79.692%.

Matricea de confuzie pe datele de validare este:

```
[[1415   58   27]
 [  76 1196  228]
 [  45  239 1216]]
```

si ne arata ca modelul nu stie sa diferentieze intre clasa 1 si 2 la fel de bine ca intre clasele 0 si 1 si 0 si 2.

References

[1] Koziarski, Michał Cyganek, Boguslaw *Impact of Low Resolution on Image Recognition with Deep Neural Networks: An Experimental Study.* International Journal of Applied Mathematics and Computer Science. 28. 735-744. 10.2478/amcs-2018-0056. 2018.