Showcase

💡 Modelul folosit pentru showcase este similar cu prima alegere de model (3DCNN). Modelul respectiv se numeste MobileNet + LSTM.

💡 Diferenta dintre approach-ul acesta si cel folosit in capitolul anterior este ca modelul acesta este multimodal. Se folosesc weight-urile modelului MobileNet, antrenat pe ImageNet (dataset cu foarte multe imagini cu foarte multe variatii de subiecte), la care se adauga un layer de LSTM, folosit pentru antrenarea pe secvente temporale.

💡 Motivul pentru aceasta schimbare arhitecturala este separarea contextelor. Lasam partea de image feature extraction sa fie abordata exclusiv de MobileNet, iar partea de secventialitate temporala este abordata de LSTM.

Dataset-ul este unul similar cu 20bnjester, avand gesturi si exemple din 20bnjester, dar se antreneaza pe o portiune mult mai mica din dataset. (Aprox 100 videoclipuri pe gest, si 100 de videoclipuri de validare).

Resurse

https://towardsdatascience.com/building-mobilenet-from-scratch-using-tensorflow-ad009c5dd42c#:~:text=Unlike normal CNN models which,refer to — Depthwise Convolutional Blocks.

https://medium.com/@godeep48/an-overview-on-mobilenet-an-efficient-mobile-vision-cnn-f301141db94d

MobileNET

MobileNET este prima retea neuronala a echipei TensorFlow in domeniul computer vision. Spre deosebire de Conv3D, MobileNet foloseste Depthwise Convolution, care este bazata pe layere de Convolutie 2D pentru care fiecare canal este procesat cu un kernel diferit.

Un Depthwise Convolution layer (MobileNet block) ar arata cam asa:

```
def mobilnet_block (x, filters, strides):

    x = DepthwiseConv2D(kernel_size = 3, strides = strides, pad
    x = BatchNormalization()(x)
    x = ReLU()(x)

    x = Conv2D(filters = filters, kernel_size = 1, strides = 1)
    x = BatchNormalization()(x)
    x = ReLU()(x)

    return x
```

In general, o schema de retea neuronala MobileNet ar arata cam asa:

Conv dw / s1	$3 \times 3 \times 32$ dw
Conv / s1	$1 \times 1 \times 32 \times 64$
Conv dw / s2	$3 \times 3 \times 64$ dw
Conv / s1	$1 \times 1 \times 64 \times 128$
Conv dw / s1	$3 \times 3 \times 128$ dw
Conv / s1	$1 \times 1 \times 128 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw
Conv / s1	$1 \times 1 \times 128 \times 256$
Conv dw / s1	$3 \times 3 \times 256$ dw
Conv / s1	$1 \times 1 \times 256 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw
Conv / s1	$1 \times 1 \times 256 \times 512$
$5\times$ Conv dw / s1 Conv / s1	$3 \times 3 \times 512$ dw $1 \times 1 \times 512 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw
Conv / s1	$1 \times 1 \times 512 \times 1024$
Conv dw / s2	$3 \times 3 \times 1024$ dw
Conv / s1	$1 \times 1 \times 1024 \times 1024$
Avg Pool / s1	Pool 7×7
FC / s1	1024×1000
Softmax / s1	Classifier

Transfer Learning

Tinand cont ca TensorFlow ofera un API prin care putem sa folosim weight-urile unui model tip MobileNet. Este necesar doar un import statement. Weight-urile fac parte din dataset-ul TensorFlow ImageNet, care a fost construit pentru object detection. Combinand ce se afla in dataset-ul nostru, putem adauga layer-ul LSTM precizat in liniile de mai sus ale documentului (Long Short Term Memory).

```
from keras.applications import mobilenet

def mobilenet_RNN(cells=128, dense_nodes=128, dropout=0.25, num_

    mobilenet_transfer = mobilenet.MobileNet(weights='imagenet',
    model = Sequential()
    model.add(TimeDistributed(mobilenet_transfer,input_shape=(nu
    model.add(TimeDistributed(BatchNormalization()))
    model.add(TimeDistributed(MaxPooling2D((2, 2))))
    model.add(TimeDistributed(Flatten()))

    model.add(LSTM(cells))
    model.add(Dropout(dropout))
    model.add(Dense(dense_nodes, activation='relu'))
    model.add(Dropout(dropout))
    model.add(Dense(num_classes, activation='softmax'))

    opt = tf.keras.optimizers.Adam()
    model.compile(opt, loss='categorical_crossentropy', metrics=
    return model
```

Dataset

Link: https://www.kaggle.com/datasets/imsparsh/gesture-recognition/data

Selectat dintr-un dataset similar cu 20bnjester, am selectat 4 gesturi, extrase din 663 de videoclipuri pentru training, si 100 de videoclipuri pentru validare. Se vor citi si transpune datele imaginilor (datele pixelilor in 3 canale) in forme de `ndarray` , conform `numpy`. Aceste date sunt normalizate (fiecare pixel value este impartit la `/255.0`).

Gesturile sunt descrise sub label-urile:

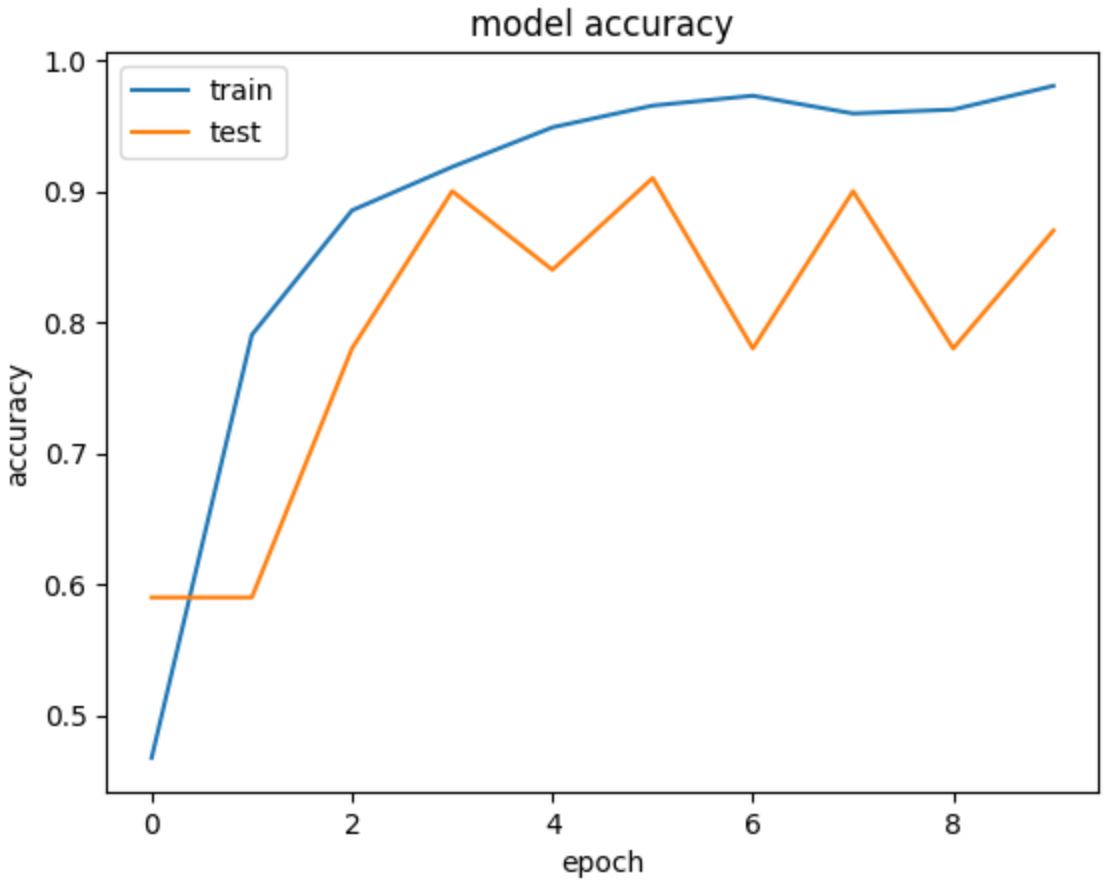
- `Swipe` (swipe left si swipe right unificat, functional swipe left) → Bind play video
- `Stop` → Bind pause video
- `Thumbs Up` → Volume up
- `Thumbs Down` → Volume down

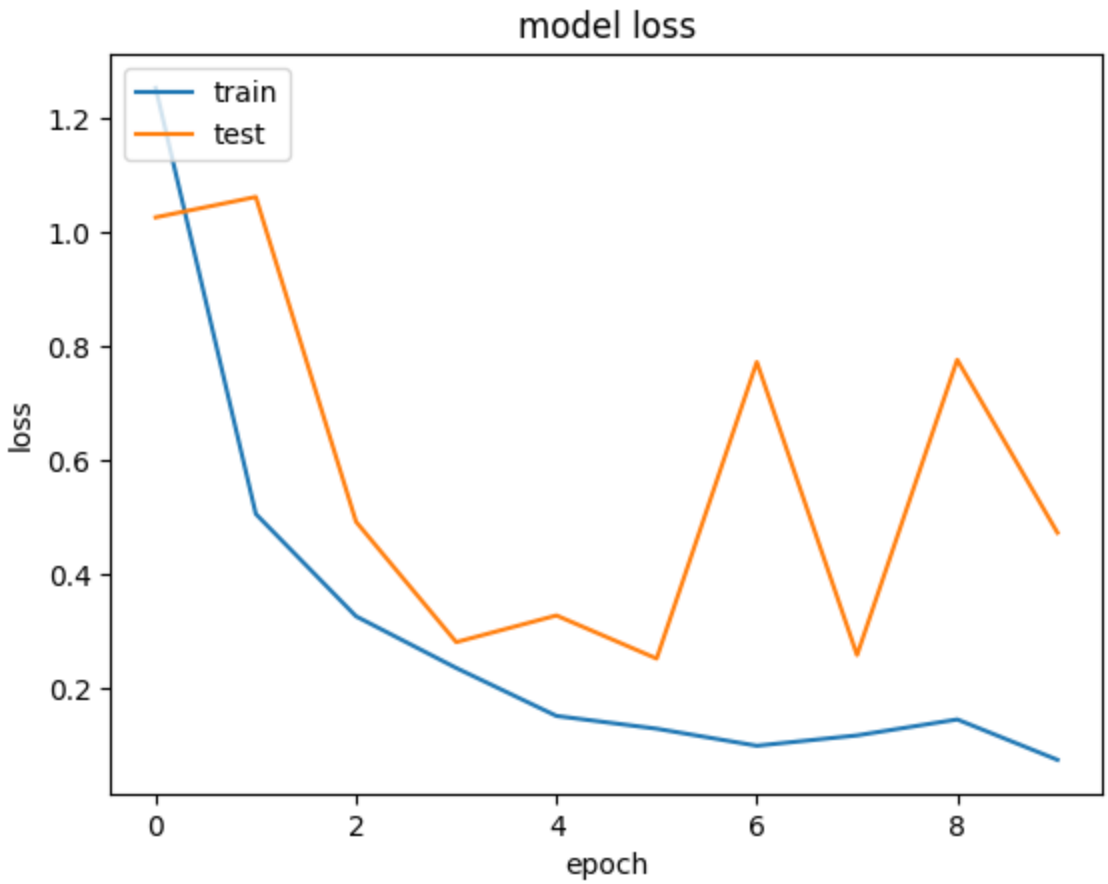
Se selecteaza 20 de frame-uri, care sunt preluate de `ImageDataGenerator` care va aplica augmentare

```
ImageDataGenerator
(
    zoom_range=0.1, # RANDOM ZOOM
    zca_whitening=True, # PIXEL DECORRELATION
    width_shift_range=0.1, # SHIFT TO UP TO 10% THE WIDTH
    height_shift_range=0.1 # ..................... HEIGHT
)
```

Training

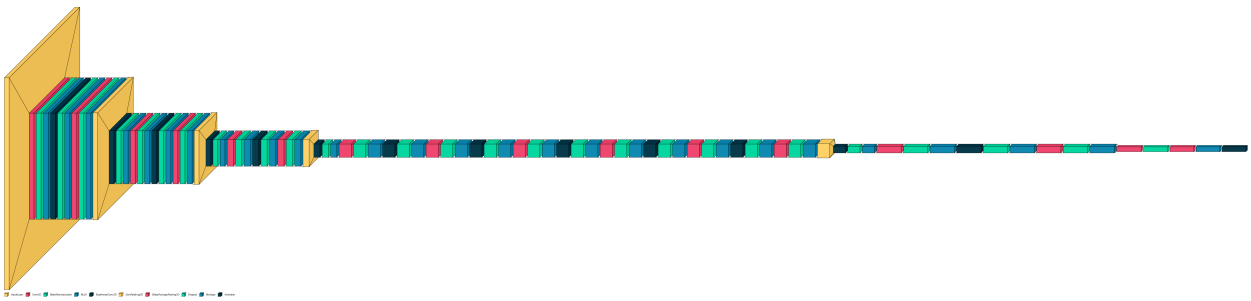
Se va antrena pe `20 epochs` , cu un `32 batch size`, cu configuratia modelului MobileNet + LSTM. Se vor aplica tehnicile de augmentare precizate. Modelul experimenteaza overfitting, dupa `epoch 7`. Acest model (comparativ cu celelalte modele precizate `3DCNN` si `MobileNet + GRU`) este cel mai bun model pentru folosinta pe webcam. Se folosesc `ModelCheckpoint` si `ReduceLROnPlateau`. Modelul in medie, bazat pe incercarea a 5 train-uri, produce 4 checkpoint-uri, dintre care penultimul are cea mai buna performanta pentru live webcam feed.


model accuracy
train
test
accuracy
epoch
1.0
0.9
0.8
0.7
0.6
0.5
0
2
4
6
8

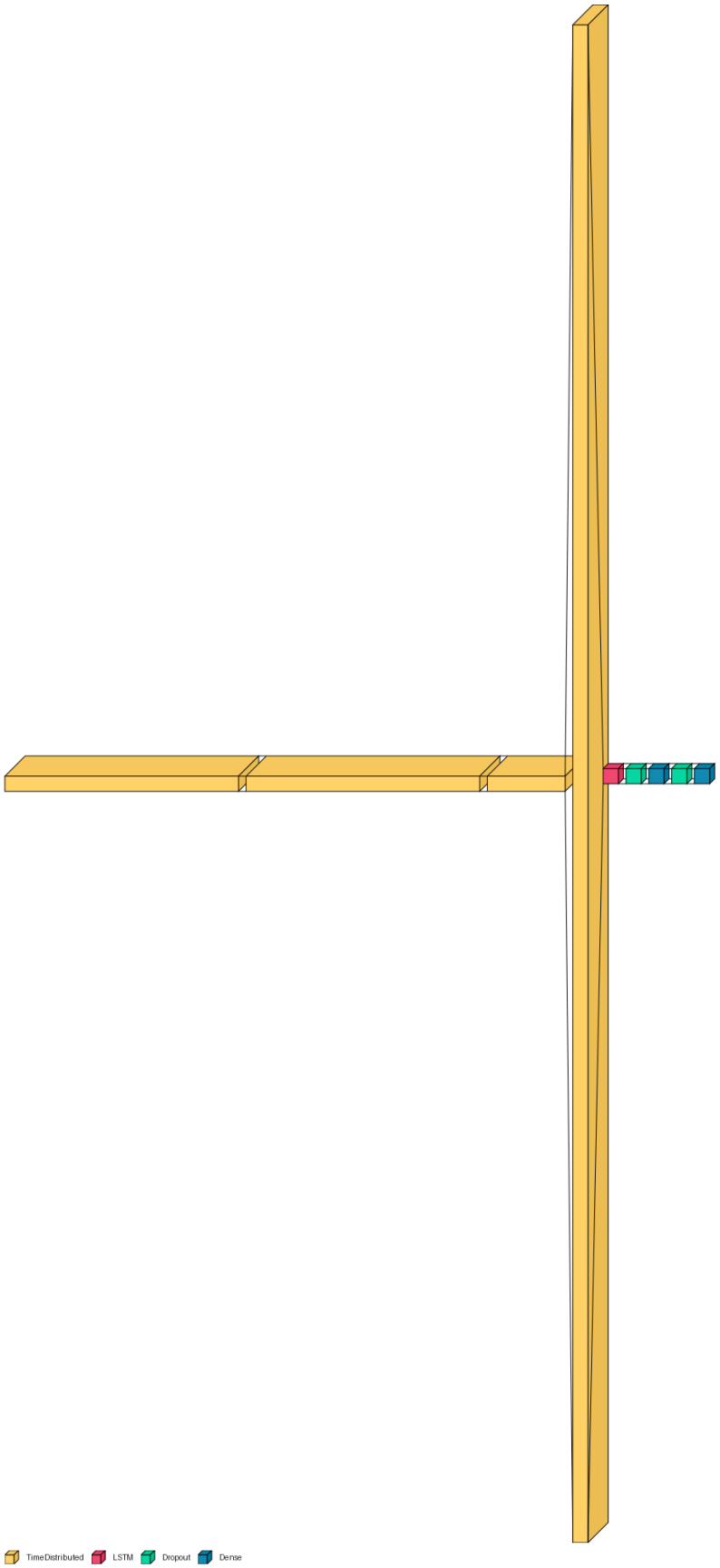


Overview

MobileNet



LSTM

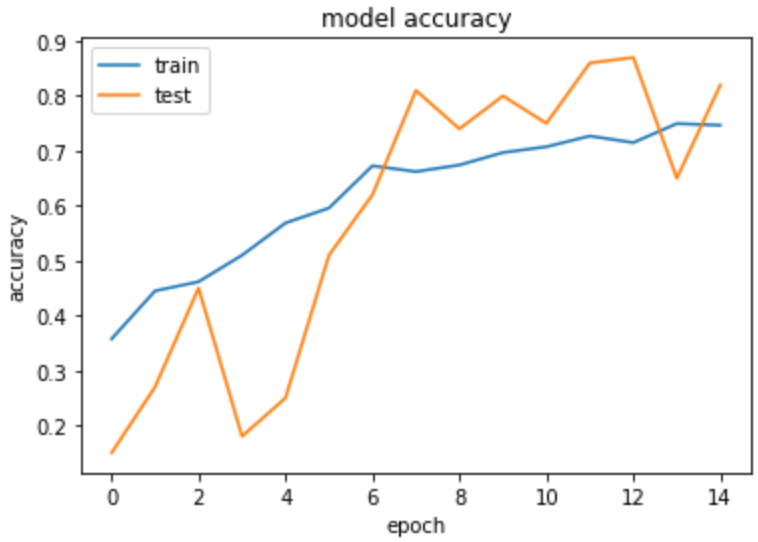

TimeDistributed
LSTM
Dropout
Dense

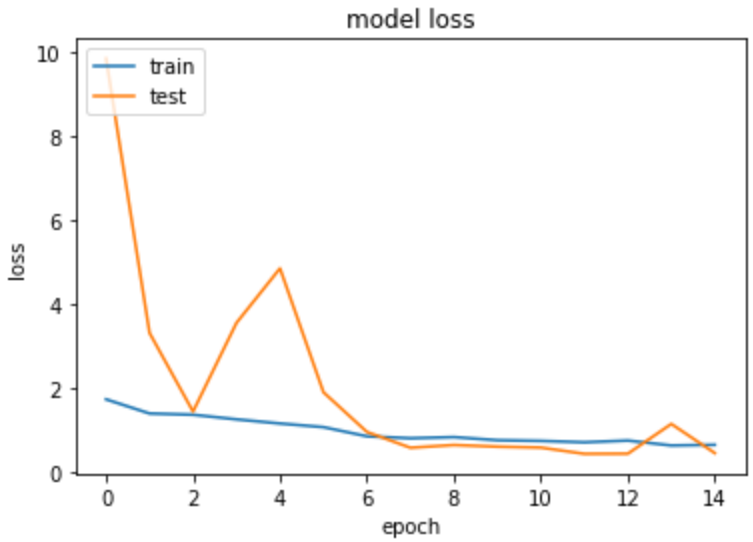
Comparatie

Modelul ales (MobileNet + LSTM) este pus in comparatie cu doua modele. Primul model are o arhitectura similara (MobileNet + GRU), iar al doilea foloseste o schema de convolutie 3D, asemanatoare celei din capitolul trecut.

Conv3D

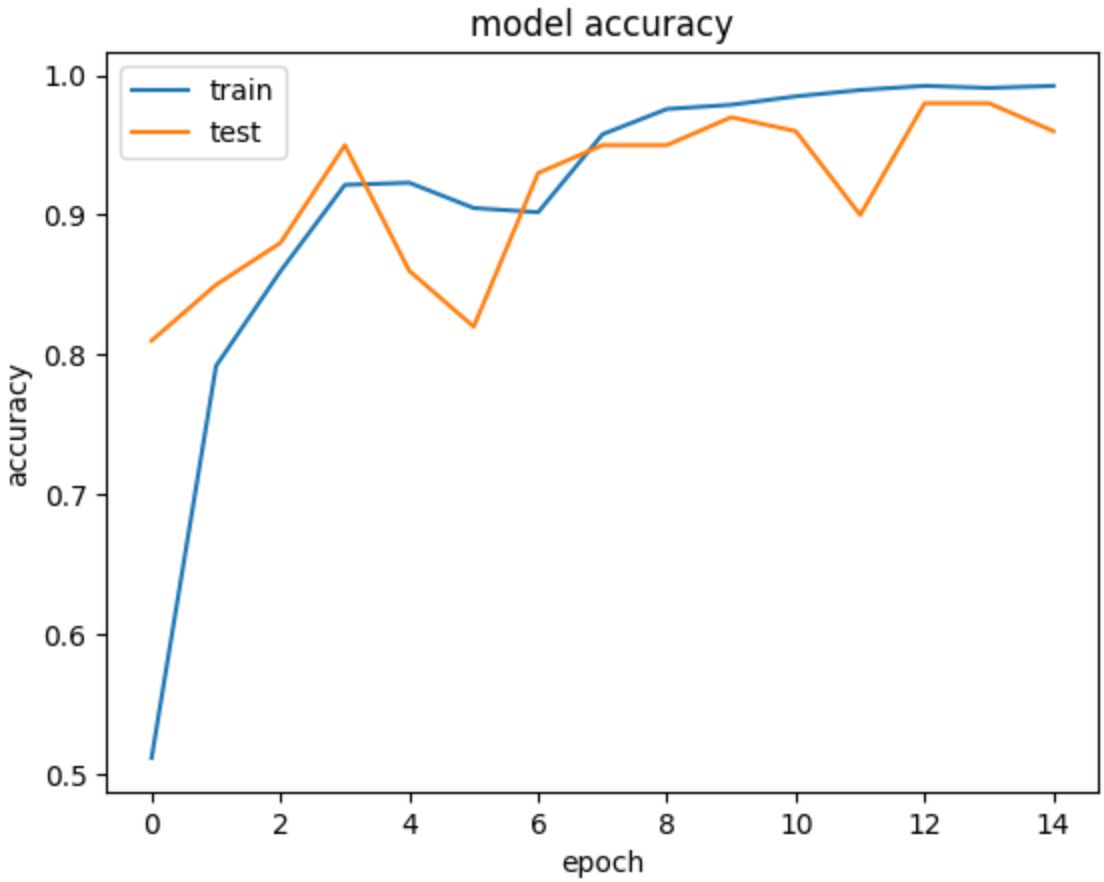
- Antrenat pe 15 epoci
- Batch size de 32
- Fara augmentare

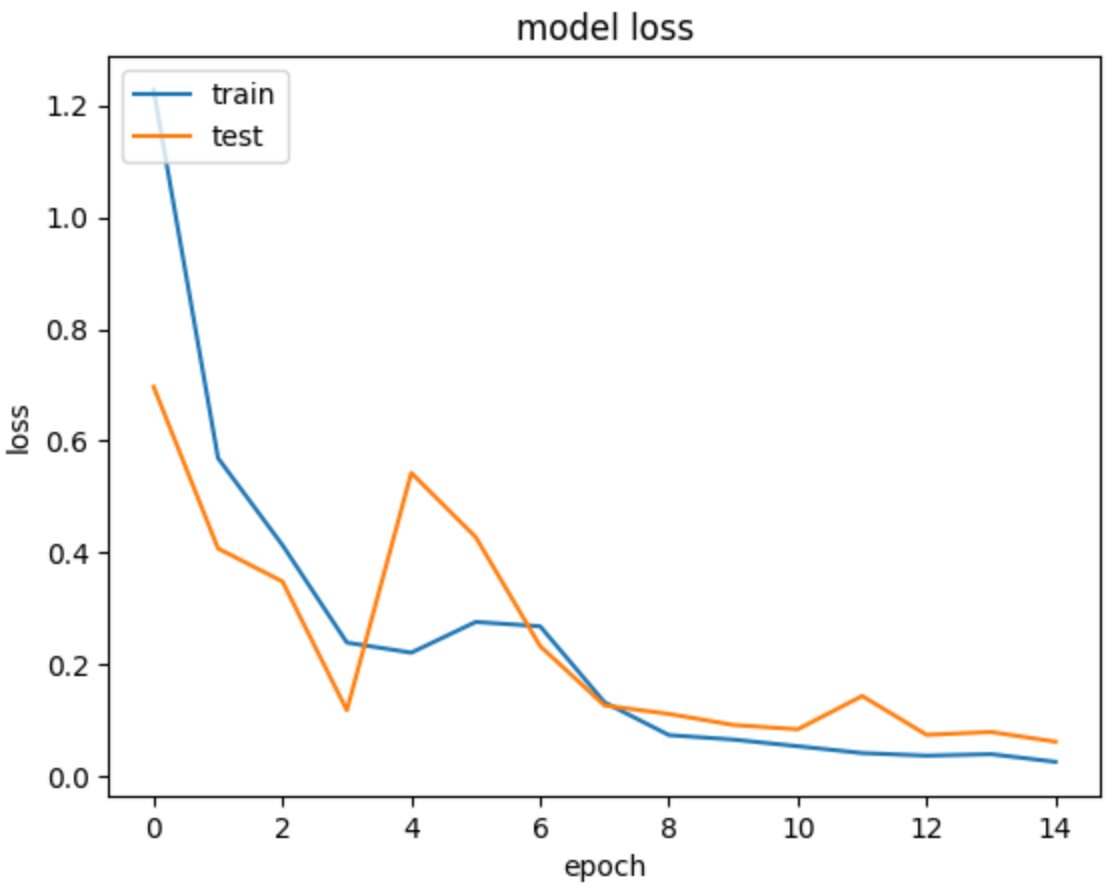




MobileNet + GRU

- Implementare similara (in loc de LSTM se folosesc GRU cells, cu acelasi numar de unitati - 128)
- Antrenat pe 15 epoci
- Batch size de 16
- Augmentare aplicata


model accuracy
train
test
accuracy
epoch
1.0
0.9
0.8
0.7
0.6
0.5
0
2
4
6
8
10
12
14


model loss
train
test
loss
epoch
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10
12
14