PRACTICE I - CONVOLUTIONAL NEURAL NETS

02/05/2021

MBD	S2
Group	G
Practice I	CNN
Date	02/05/2021

ToC

Introduction

Goals

As described in group practice document:

- To be familiar with convolutional neural nets as a powerful supervised machine learning classification technique.
- To be familiar with TensorFlow & Keras as frameworks to develop deep neural networks.
- Understand how to regularize a neural net.
- Use Transfer Learning to improve the accuracy of your net.

Report

In the present report we, group G, detail answers, explanations and snippets of code (when needed) required to respond to the questions and tasks specified in the group practice document. For a complete view of the process and code we kindly refer the reader to the output notebook(s) specified for each section.

Sections

Section 1: Cats & Dogs

Notebooks

Input: 01_cnn_template.ipynb

Output: 01_Cat_vs_Dog_Image_cls_done.ipynb

Exercise 1.1

Create the second convolution layer followed by a MaxPooling2D layer

In order to create a convolutional layer, we first checked the official Keras Doc. Here we can find the parameters that we are looking for (filter, kernel-size and activation function) `x = layers.Conv2D(filters, kernel_size, activation)(input)`

For the MaxPooling2D layer we just had to add the pool_size. Since we did not specifiy any stride, keras automatically defaults `stride` to the pool_size value.

Solution

```
# Second convolution extracts 32 filters that are 3x3
# Convolution is followed by max-pooling layer with a 2x2 window

x = layers.Conv2D(32, 3, activation='relu')(x)

x = layers.MaxPooling2D(2)(x)
```

Exercise 1.2

Flatten the output of the precedent layer, resulting in a 1D layer

Again, the Keras Documentation for layer.Flatten() helped us to find the appropriate values to achieve that.

Solution

```
x = layers.Flatten()(x)
```

Exercise 1.3

Create the fully connected layer(dense)

Create a fully connected layer with ReLU activation and 512 hidden units

Solution

```
x = layers.Dense(512, activation='relu')(x)
```

Section 2: Regularization

Notebooks

Input: 02_cnn_template.ipynb

Output: 02_Cat_vs_Dog_regularization_done.ipynb

Exercise 2.1

Data Augmentation. Explain how it is working ImageDataGenerator. Specifically, explain what all the parameters, already set, and their respective values mean. One by one, from rotation_range to fill_mode.

ImageDataGenerator is used to generate batches of tensor image data with real-time data augmentation. It takes images and alters them according to the parameters passed to it.[1] This way it is possible to randomly create more training data, which is especially useful when the original dataset is small.

Parameters:[2]

concept	definition
rotation_range	sets the random rotation_range
width_shift_range	sets a range for horizontal shifts
height_shift_range	sets a range for vertical shifts
shear_range	sets a range for counter clockwise shearing (degrees)
zoom_range	sets a range for zoom
horizontal_flip	boolean value indicating if the picture can be flipped at random
fill_mode	Points outside the boundaries of the input are filled according to the given mode

Exercise 2.2

Explain Dropout as a regularization technique

To avoid overfitting, Dropout is used to randomly set input units to zero each update cycle. This allows for models to be more robust, better at generalization and less likely to overfit. Setting the attribute to 0.5 means that we are setting half of our input units to 0. For reproducibility, we used the seed parameter too.

---

[1] More information can be found in keras documentation

[2] Extracted from this blog

Add a Dropout layer with a dropout rate of 0.2.

```
x = layers.Dropout(rate=0.2, seed=123)(x)
```

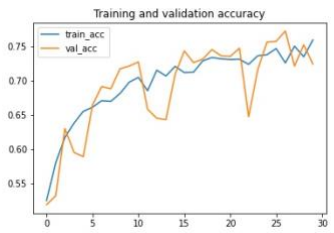
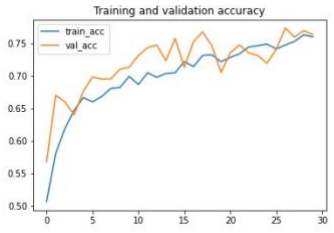
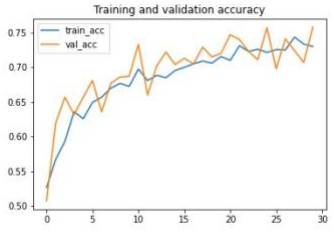
Write code to train the model on all 2000 images for 30 epochs and validate on all 1000 validation images.

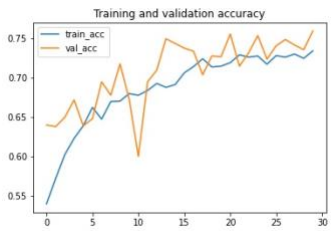
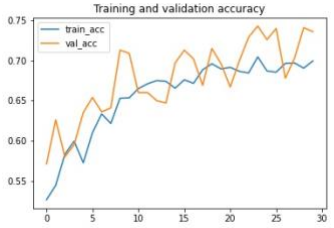
Solution

Here, we were making use of the code from the previous exercise. We only had to change the epochs, since the batch-sizes and steps stay the same.

```
history = model.fit(
    train_generator,
    steps_per_epoch=100, # 2000 images = batch\_size \* steps
    epochs=30,
    validation_data=validation_generator,
    validation_steps=50, # 1000 images = batch_size * steps
    verbose=2)
```

Try dropout of 0.9 and 0.1 rate. Explain the different behavior.

dropout rate	accuracy	loss
no dropout		
0.1		
0.2		

0.5		
0.9		

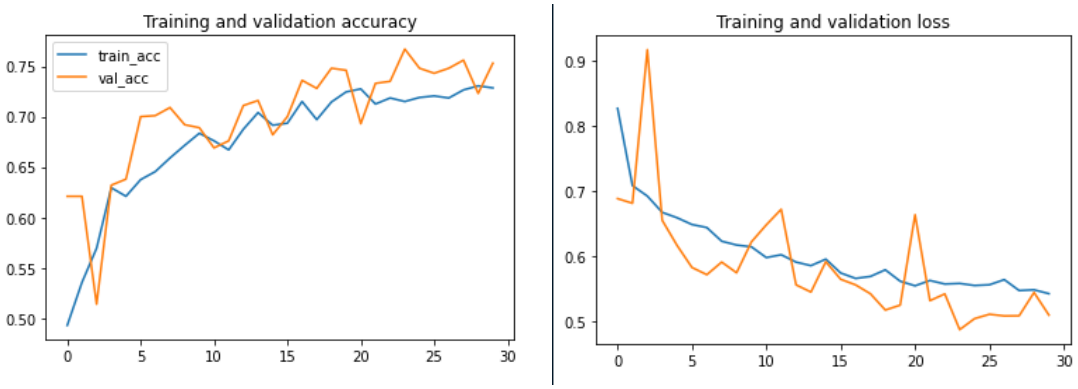
Here we can see, how the different dropout-rates affect the training and inference of these models. With 30 epochs we observe the following,

- When the dropout rate is high enough (above 0.5), we can see that the performance to unseen data is a bit erratic. This erratic behavior is lower with 0.1-0.2 rates.
- Also, with no dropout rate the expectation would be for the model to overfit over models (NN) with a dropout rate with a higher number of epochs.

Exercise 2.3

Fit the final model following the specifications in the code

Below the resulting accuracy and loss from the train and validation steps.



Section 3: Stand on the shoulders of giants

Notebooks

Input: 03_cnn(1).ipynb, 04_cnn_template(1).ipynb

Output: 03_feature_extraction_fine_tunning_done.ipynb, 04_cnn_from_scratch_optimized_done.ipynb, 04_cnn_template(1)_models.ipynb, 04_cnn_template(1)_models(2).ipynb

Exercise 3.1

Do some research and explain the inception V3 topology. Which database do they use for training it? How was trained?

Inception V3

Note: Inception V4 is the latest version. This version brings a simplified architecture and more modules. An overall Explanation and list of features can be found in the provided links.

What is Inception V3?

- A convolutional neural network (CNN) used for image analysis and object detection
- Part of the Inception family, which first started as a module of GoogLeNet. For more information about GoogLeNet see (Szegedy et al., 2014).
- V3 introduces ways to factorize convolutions with large filter size (e.g. factorized 7 x 7 convolutions), applies aggressive regularization via label smoothing (Szegedy et al., 2015).[3]

Topology

Inception V3 uses the architecture depicted in the table below.

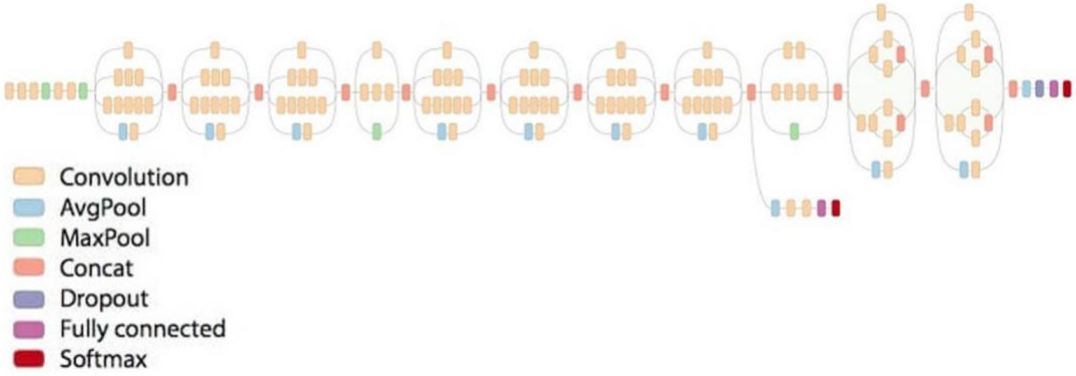
type	patch size/stride or remarks	input size
conv	3×3/2	299×299×3
conv	3×3/1	149×149×32
conv padded	3×3/1	147×147×32
pool	3×3/2	147×147×64
conv	3×3/1	73×73×64
conv	3×3/2	71×71×80
conv	3×3/1	35×35×192
3×Inception	See reference	35×35×288
5×Inception		17×17×768
2×Inception		8×8×1280
pool	8 × 8	8 × 8 × 2048
linear	logits	1 × 1 × 2048
softmax	classifier	1 × 1 × 1000

V3_topology

[3] Szegedy et al. describe *label smoothing* as a "mechanism to regularize the classifier layer by estimating the marginalized effect of label-dropout during training"

source: (Szegedy et al., 2015)

A high-level view of the topology is provided in the figure below.

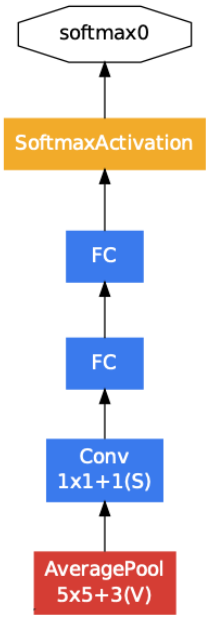


V3_high-level

source: ("Inception V3 Deep Convolutional Architecture For Classifying Acute...," n.d.)

Components

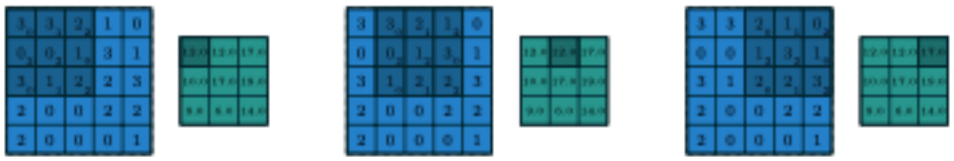
- Auxiliary classifier: architectural component used with the goal to improve convergence during training (i.e. find suitable solutions).



aux-classifier

source: (Szegedy et al., 2014)

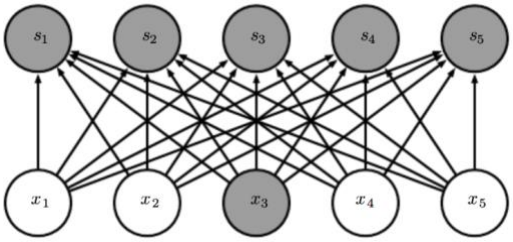
- Convolution: matrix operation that consists of applying a kernel (a.k.a. filter) sliding over data performing element-wise multiplication summing out the output.



convolution

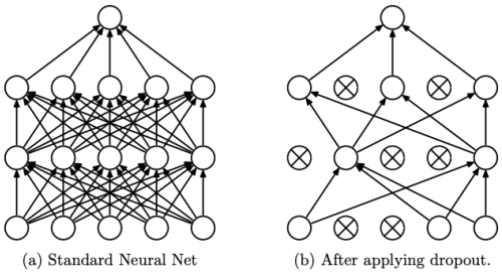
source: (Dumoulin and Visin, 2018)

- Dense connections (feedforward networks): layer where every input is connected to every output by a weight.



Dense layer

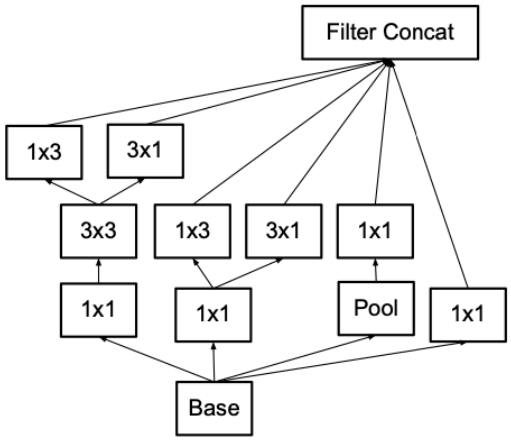
- Dropout (regularization): regularization technique that drops a unit and connections. This is done while training. The probability for dropping units can be set.



dropout

source: (Srivastava et al., n.d.)

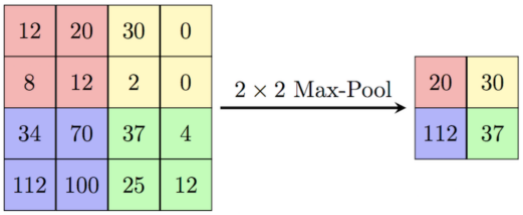
- Inception-V3 module (image model blocks): image block used to promote high dimensional representations.



inception_module

source: (Szegedy et al., 2015)

- Label smoothing (regularization): technique that introduces noise for the labels.
- Max pooling: operation that gets the maximum value of the sub-sample of a feature map creating a smaller feature map.



max_pooling

source

- Softmax (output function): function that transforms input (last's layer output) into a vector of probabilities.

Training database

To train the third version of inception a dataset of 1000 classes was used from the original ImageNet dataset ("Inception V3 Deep Convolutional Architecture For Classifying Acute...," n.d.).

Training approach

According to (Szegedy et al., 2015), Inception V3 was trained using the following:

Concept/Parameter/Hardware	Value
Solver	Stochastic gradient
ML system	TensorFlow distributed machine learning system

Replicas	50
Hardware	NVidia Kepler GPU (per replica)
Batch size	32
Epocs	100
Learning rate	0.045

Exercise 3.2

Inception V3 is set by default to admit input images of (299, 299, 3) dimensions; but we want (and we will use) inputs of (150, 150, 3). If the net is already trained, how is that even possible?

Note that when applying transfer learning, using images of sizes different than the one used for training (as done in this exercise) may have an impact on the performance of the neural net. Although, using as input images with a lower size could reduce training time of the "new" neural network as discussed in this post.

Inception V3 needs to receive three (3) color channels - depth of the image. Using input images of a lower size than the one used for training does not change the number of channels but the size of the feature map[4].

Exercise 3.3

Let's change the database. Will Inception work well with different objects? Use Inception V3 to outperform the results of a small convnet in a flower database. Complete the code in 04_cnn_template.ipynb to use Inception V3 in this database in the same way we did it for cats & dogs.

Model definition

For this exercise we have defined several scenarios. In the table below we present the characteristics for each scenario. Note that the scenarios selected represent a sub-set of the potential combinations of characteristics.

Models	Data augmentation	Regularization (weights)	Optimizer	Inception V3
Baseline	No	No	RMSprop	No
M1	No	Yes	Adam	No
M2	No	Yes	Adam	Yes

[4] Also known as activation map is the output of applying a kernel to an image (activations of a filter). source

M3	Yes	No	RMSprop	No
M4	Yes	Yes	Adam	No
M5	Yes	Yes	Adam	Yes

All models have the following characteristics:

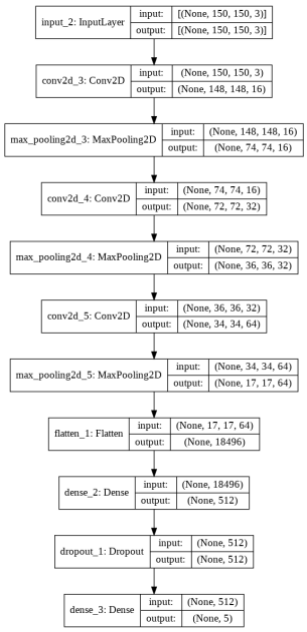
- Dropout layer (seed = 42)
- Five (5) nodes in the output layer
- Softmax activation function in the output layer and ReLU in dense layers
- L1 & L2 weight regularizers (l1=1e-5, l2=1e-4)
- Dropout rate of 0.2 (models baseline, M1 and M2); with data augmentation (models M3, M4 and M5) is 0.5
- 100 epocs, but using early stopper

We use Layer weight regularizers to apply penalties on layer's kernel. According to this post it is recommended to use low values for L1 and L2.

As explained in exercise 2.1, the dropout rate specifies the fraction of the input units to drop.

Structure

Below the structure of the neural network without Inception V3.

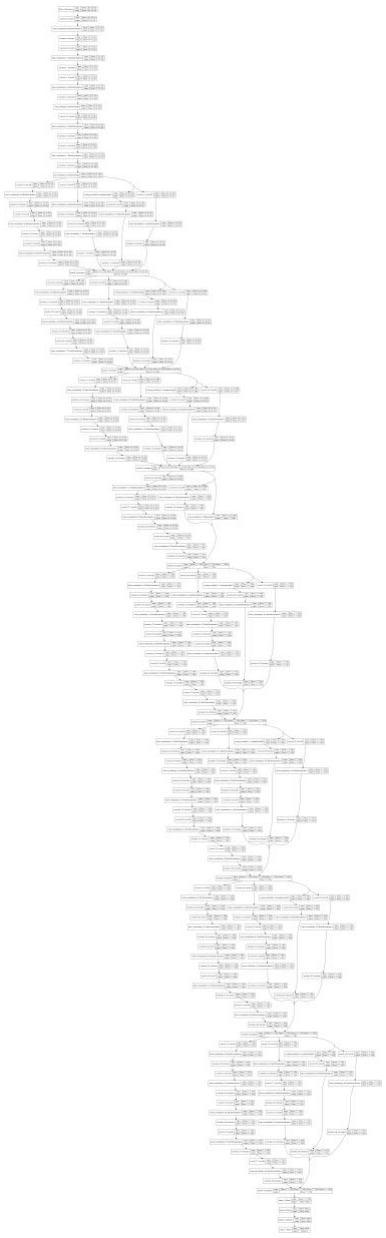


model structure without Inception V3

```
Model: "baseline & M1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
```

```
input_2 (InputLayer)         [(None, 150, 150, 3)]     0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 148, 148, 16)      448
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 74, 74, 16)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 72, 72, 32)        4640
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 36, 36, 32)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 34, 34, 64)        18496
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 17, 17, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 18496)             0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               9470464
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_3 (Dense)              (None, 5)                 2565
=================================================================
Total params: 9,496,613
Trainable params: 9,496,613
Non-trainable params: 0
```

Below the structure of the neural network with Inception V3.



model structure with Inception V3

Below a snapshot of the last layers. *Note* that Inception V3 layers not shown refer to pre-trained layers.

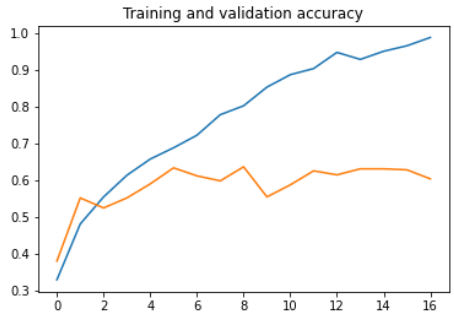
```
Model: “baseline & M1"****
__________________________________________________________________________________________
____
Layer (type)                    Output Shape         Param #     Connected to
==========================================================================================
====
__________________________________________________________________________________________
____
activation_539 (Activation)     (None, 7, 7, 192)    0           batch_normalization_539[0][0]
__________________________________________________________________________________________
____
```

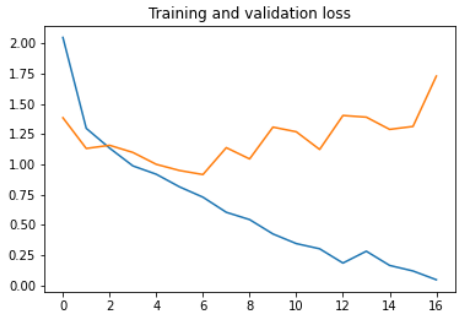
```
mixed7 (Concatenate)            (None, 7, 7, 768)    0           activation_530[0][0]
                                                                 activation_533[0][0]
                                                                 activation_538[0][0]
                                                                 activation_539[0][0]
__________________________________________________________________________________________
____
flatten_22 (Flatten)            (None, 37632)        0           mixed7[0][0]
__________________________________________________________________________________________
____
dense_44 (Dense)                (None, 1024)         38536192    flatten_22[0][0]
__________________________________________________________________________________________
____
dropout_13 (Dropout)            (None, 1024)         0           dense_44[0][0]
__________________________________________________________________________________________
____
dense_45 (Dense)                (None, 5)            5125        dropout_13[0][0]
==========================================================================================
====
Total params: 47,516,581
Trainable params: 38,541,317
Non-trainable params: 8,975,264
```

Training and evaluation

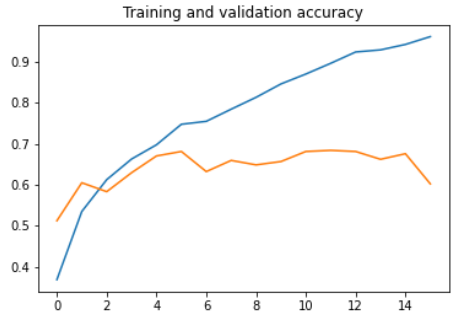
Below we present the accuracy and loss on the training and validation sets for each model.

Baseline

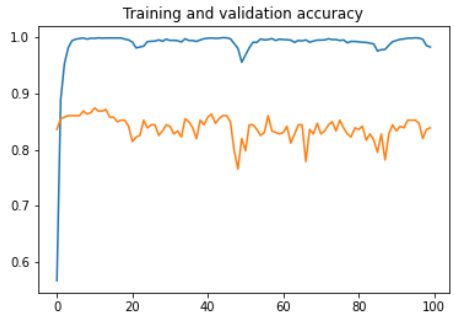


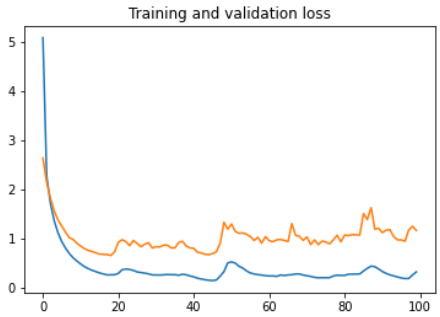


M1

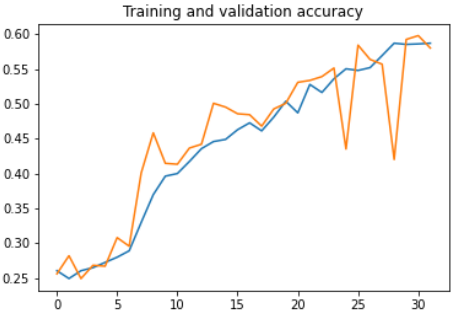


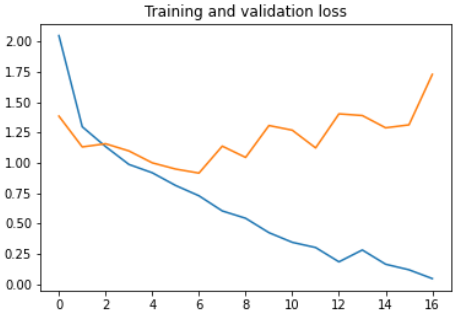
M2


Training and validation accuracy
1.0
0.9
0.8
0.7
0.6
0
20
40
60
80
100

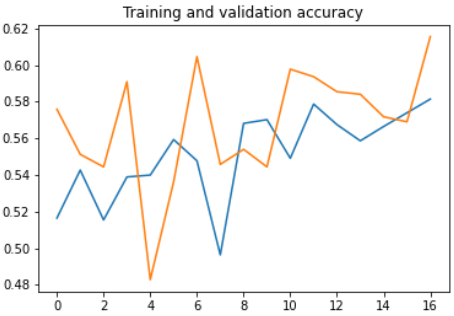

Training and validation loss
5
4
3
2
1
0
0
20
40
60
80
100

M3


Training and validation accuracy
0.60
0.55
0.50
0.45
0.40
0.35
0.30
0.25
0
5
10
15
20
25
30

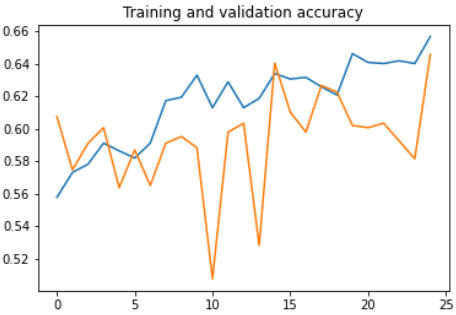

Training and validation loss
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
2
4
6
8
10
12
14
16

M4


Training and validation accuracy
0.62
0.60
0.58
0.56
0.54
0.52
0.50
0.48
0
2
4
6
8
10
12
14
16


Training and validation loss
1.35
1.30
1.25
1.20
1.15
1.10
0
2
4
6
8
10
12
14
16

M5


Training and validation accuracy
0.66
0.64
0.62
0.60
0.58
0.56
0.54
0.52
0
5
10
15
20
25


Training and validation loss
1.5
1.4
1.3
1.2
1.1
1.0
0
5
10
15
20
25

From the results above we observe that M2 performs better than the other 5 models. The fact that M2 seems to outperform models with data augmentation was unexpected.

Future work/fine-tuning

- Exploring different configurations with models that have data augmentation.
- Test different dropout rates.
- Test different tolerances of the early stopper.

Section 4: Object Detection - YOLO nets

Notebooks

Input: yolo.ipynb

Output: yolo.ipynb

Exercise 4.1

Describe YOLO neural net

YOLOv5 (You only look once) is a real-time object detection model introduced in 2016 by collaborators from University of Washington, Allen Institute for AI and Facebook AI Research in the paper (Redmon et al., 2016).

The model is implemented as a convolutional neural network and optimized for fast operating speed and parallel computations.

It can be executed on conventional Graphic Processing Units (GPUs) and will achieve real-time, high-quality results.

The different configurations of the layers can be found in /yolov5/models/*.yaml. It is important to adapt these configurations accordingly to the training-data, otherwise the accuracy will be very poor, or the training would result in an error.

Yolo v3 was the first stable and broadly used version of Yolo, with v4 many optimizations were introduced and with the final Version 5 now the whole model is even more streamlined, and more accessible than ever.

Exercise 4.2

Put to work YOLO_v5 in the database you want and send to me the notebooks with your comments

For this exercise, we are using YOLO v5 and the dataset comes from roboflow.

Our neural network can identify 7 classes of sea animals (fish, jellyfish, penguin, puffin, shark, starfish and stingray).

References

Dumoulin, V., Visin, F., 2018. A guide to convolution arithmetic for deep learning. ArXiv160307285 Cs Stat.

Inception V3 Deep Convolutional Architecture For Classifying Acute... [WWW Document], n.d. . Intel. URL https://www.intel.com/content/www/us/en/develop/articles/inception-v3-deep-convolutional-architecture-for-classifying-acute-myeloidlymphoblastic.html (accessed 4.27.21).

Redmon, J., Divvala, S., Girshick, R., Farhadi, A., 2016. You Only Look Once: Unified, Real-Time Object Detection. ArXiv150602640 Cs.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R., n.d. Dropout: A Simple Way to Prevent Neural Networks from Overfitting 30.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A., 2014. Going Deeper with Convolutions. ArXiv14094842 Cs.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z., 2015. Rethinking the Inception Architecture for Computer Vision. ArXiv151200567 Cs.