

NNDL@UNIPD

# ENSEMBLING CNN MODELS FOR 3D OBJECT CLASSIFICATION BASED ON VOXEL GRID REPRESENTATION

## Authors

D'Antimo Simone<sup>1</sup>  
Singh Harjot<sup>1</sup>

simone.dantimo@studenti.unipd.it  
harjot.singh@studenti.unipd.it

<sup>1</sup>The University of Padua

## Ensembling CNN models for 3D Object Classification based on Voxel Grid representation

Simone D'Antimo<sup>†</sup>, Harjot Singh<sup>†</sup>, Jacopo Pegoraro<sup>†</sup>

**Abstract**—Convolutional Neural Networks (CNN) have been widely used as Deep learning tools to enhance the spatial relationship among images and videos. In this paper, we illustrate the potential of CNN and Voxel grids for the 3D Object classification task. Intuitively, given a 3D image, we split it into blocks, each of those blocks may be only full or empty, which implies that the resolution of the 3D object is strongly correlated with the number of blocks in which they are split in, as trade-off more blocks implies more memory required for storage. We decided to evaluate the potentiality and the limits of a 3D Object Classification System having a limited amount of computational capacity and memory storage: those limitations imply that we must use samples with a lower resolution (i.e. composed by a limited number of voxel), losing some details and shape of the objects, but at the same time the results obtained show that we created a robust model which perform efficient analysis and can classify the objects independently of their rotation. We were able to reach good results on the test set, with an accuracy rate of 88 percent at the end.

**Index Terms**—Neural Networks, Convolutional Neural Networks, 3D Object Classification, Voxel Grid, ModelNet10, Data Augmentation.

### I. INTRODUCTION

The 3D Object recognition task contributes to the field of computer vision by enabling the detection, classification and also localization of three-dimensional objects in images or video streams. Thanks to the improvements in Computer Vision and Deep Learning technologies these tasks are more and more reliable and accessible, enabling many emerging applications like augmented reality, virtual reality and many more related to robotics interactions with the environment. The way 3D data is represented has a significant impact on the performance of the 3D object detection. Here we introduce two of the most common representation for 3D Objects.

- The Point-based representation uses a set of 3D points to describe the geometry of an object. This representation is lightweight and can handle large point clouds efficiently. The structure can retain precise point positions, but at the same time it has a limited feature representation and since the points are unordered using them requires a high computational capacity [1].
- Voxel-based representation models the object as a 3D

memory, since we have to store also the empty blocks, therefore they are not scalable as other approaches [1].

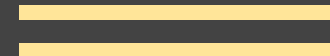
In this paper, we propose a "label predictor" that relies on 5 models using an ensembling technique to classify 3D objects, using at the same time a coarse voxel granularity. As mentioned above, the Voxel Grid structure maintains information regarding the shape and structure of the 3D Object, in order to properly capture and maximize the spatial relationship provided by the Voxel-Grid structure, we decided to use CNNs as core networks.

The main advantages of using this kind of approach is that each weak model is trained on a portion of the dataset, requiring smaller and thus quicker models. Since the weak models are trained on different portion of the dataset, they learn and make prediction based on different features, increasing the generalization of our ensemble system. If one can run all 5 weak models in parallel, and then choose the final label, this may result faster than running a single bigger model.

The memory requirements for voxelized 3D objects increase cubically with the resolution. We have seen that, for a resolution of 16, where each object is divided in a 16x16x16 voxel grid, the visualization is too coarse to be used. On the other hand, a resolution of 64 requires 262,400 blocks for each object, which is too much considering our memory capacity. In the end, we went with the resolution of 32x32x32, which has an affordable amount of blocks per objects (~32,000) [1]. In our case, this resolution is the optimal trade-off between memory and information loss. The expectation is that using a restrained number of blocks for each Object, the model will be more efficient by taking less time to execute and requiring less memory to storage all the Voxels information.

The dataset which we used for training our models is a modified version of the ModelNet10 [2]. The initial dataset is composed by 4899 aligned 3D CAD model divided in 10 different classes, we perform data augmentation to create a portion of rotated entries, which is useful to make our models learn how to classify objects independently of their rotation in the Z axis.

The rest of the paper is organized as follows: Section II provides an overview of related works; Section III describes the dataset and the experimental setup; Section IV presents the results and the discussion; Section V summarizes the conclusions.



# TABLE OF CONTENTS

The Challenge	03
Our Approach	05
Data preparation	07
Model	13
Training & Experimental results	15
Conclusions and Future work	20

# THE CHALLENGE



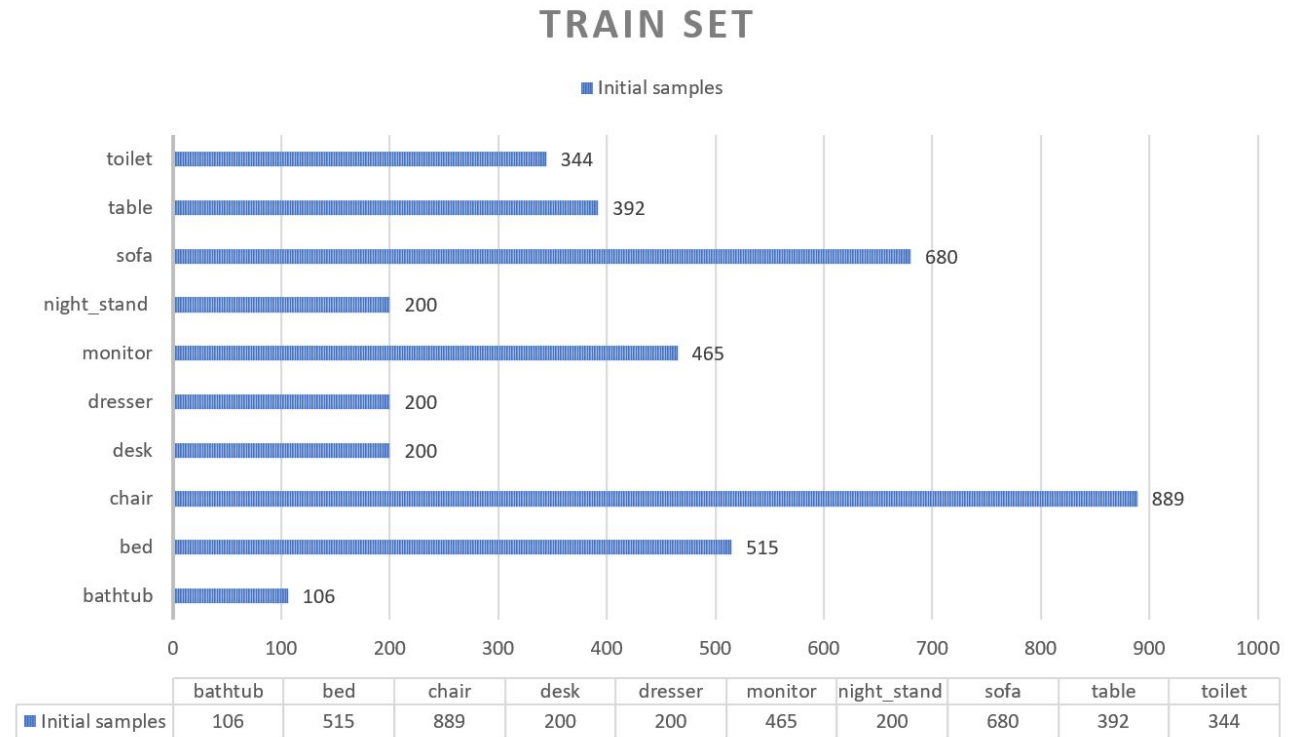
# THE CHALLENGE

**Our goal:** Classifying 3D Object independently their rotation using CNN and Voxel Grid representation

**Our data:** ModelNet10,

- 4899 3D CAD divided into 10 classes
- 80/20\* = train/test\* splitting
- orientation aligned

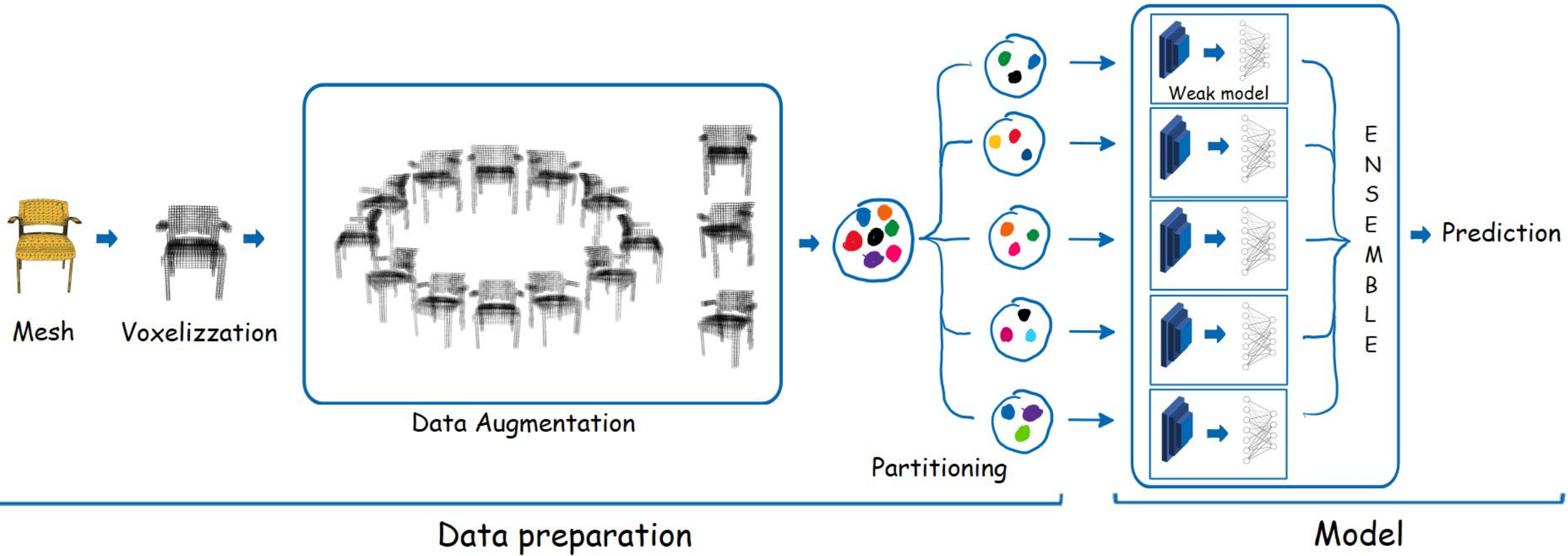
- CAD models
- Non i.i.d.
- Desired partition 80-10-10



# OUR APPROACH



# PROCESSING PIPELINE



# DATA PREPARATION



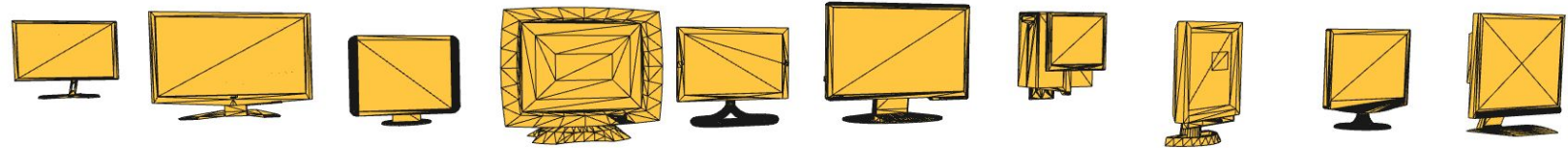


# DATA PREPARATION

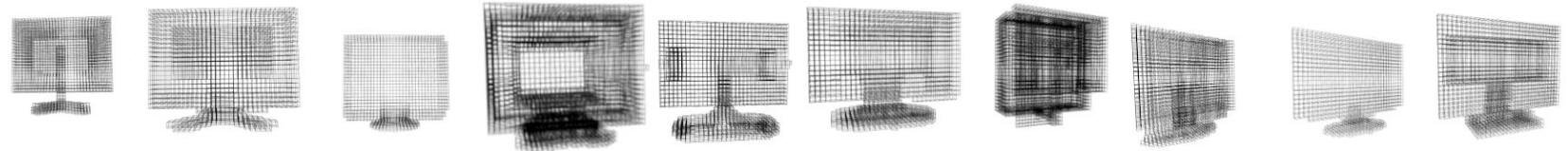
## VOXELIZATION

1

ModelNet10  
3D Cad Models



Voxel Grids





# DATA PREPARATION

VOXELIZATION

>

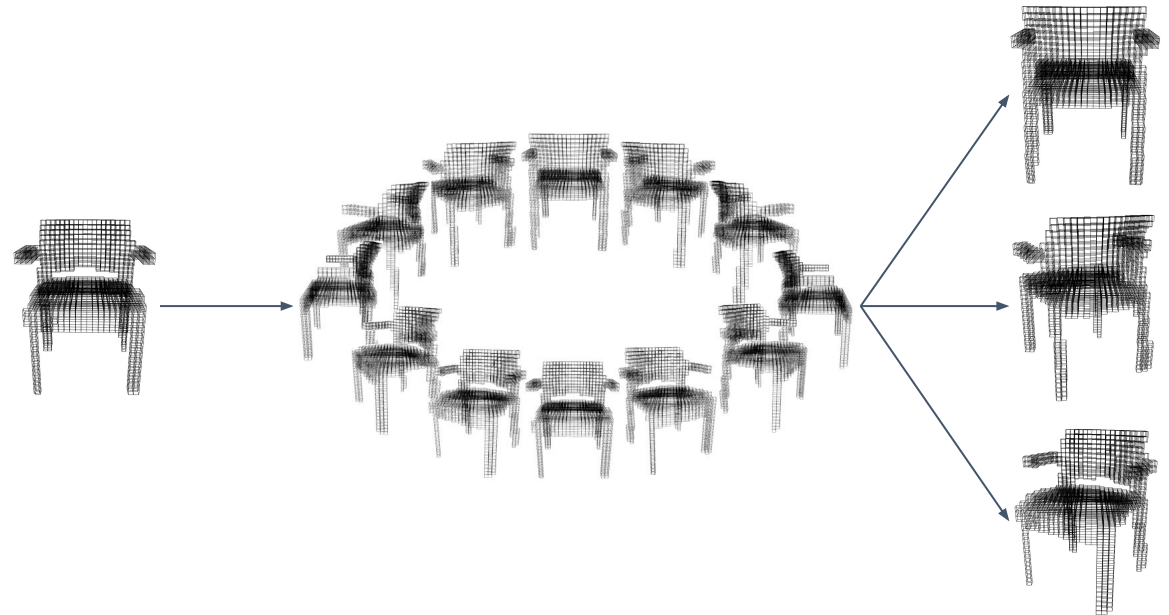
**AUGMENTATION**

>

1 **Focus** on making the **dataset IID**

2 **N Rotations**, depending on their initial distribution

Rotations	Classes
2	bed, chair, monitor, sofa
3	table, toilet
5	desk, dresser, night_stand
8	bathtub



# DATA PREPARATION

VOXELIZATION

>

AUGMENTATION

>

1 Focus on making the dataset IID

2 N Rotations, depending on their initial distribution

Rotations	Classes
2	bed, chair, monitor, sofa
3	table, toilet
5	desk, dresser, night_stand
8	bathtub



# DATA PREPARATION

VOXELIZATION

>

AUGMENTATION

>

**PARTITIONING**

1

Focus on making the **dataset** ready for the **ENSEMBLE** models

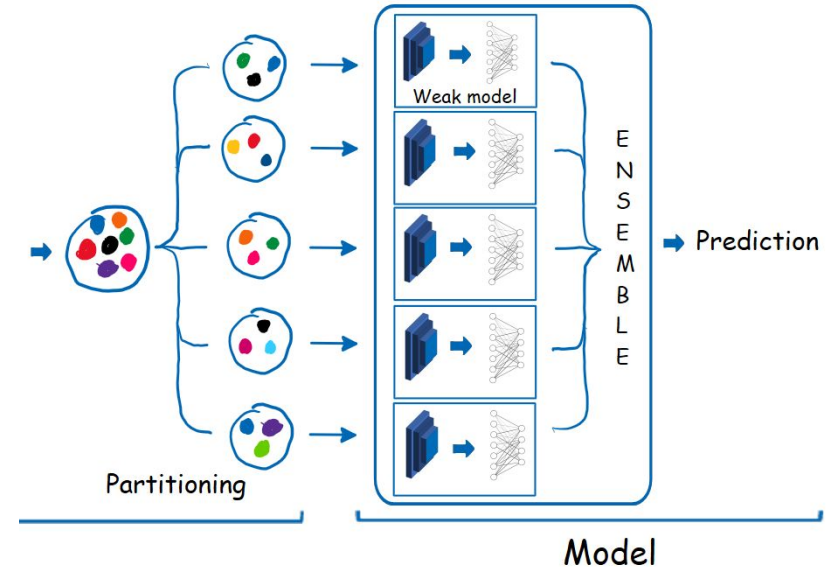
2

Each weak model should learn

- well on their part of the dataset
- some information on data of other models.

TABLE 2: Our Bagging datasets compositions

Dataset	Chunk's subsets	Copied Subset
1	01, 02, 03, 04, 05	10, 15, 20, 25 — 24
2	06, 07, 08, 09, 10	01, 11, 16, 21 — 05
3	11, 12, 13, 14, 15	02, 07, 17, 22 — 06
4	16, 17, 18, 19, 20	03, 08, 13, 23 — 12
5	21, 22, 23, 24, 25	04, 09, 14, 19 — 18



# DATA PREPARATION

VOXELIZATION

>

AUGMENTATION

>

PARTITIONING

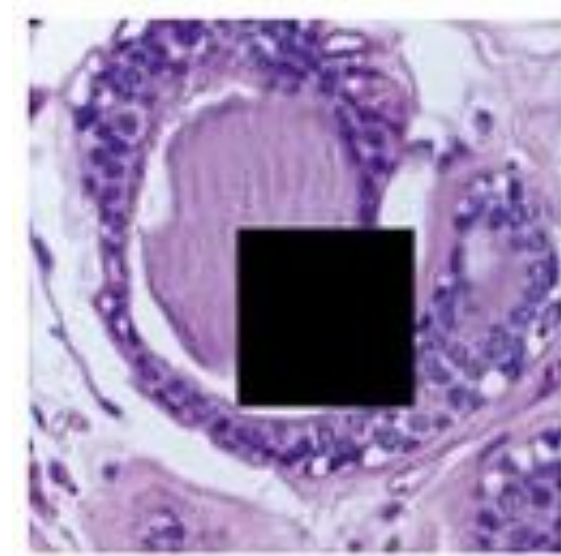
>

**CUTOUT**

1

**Masking Out** a random portion of the input data

- Improve robustness
- Improve generalization



Cutout

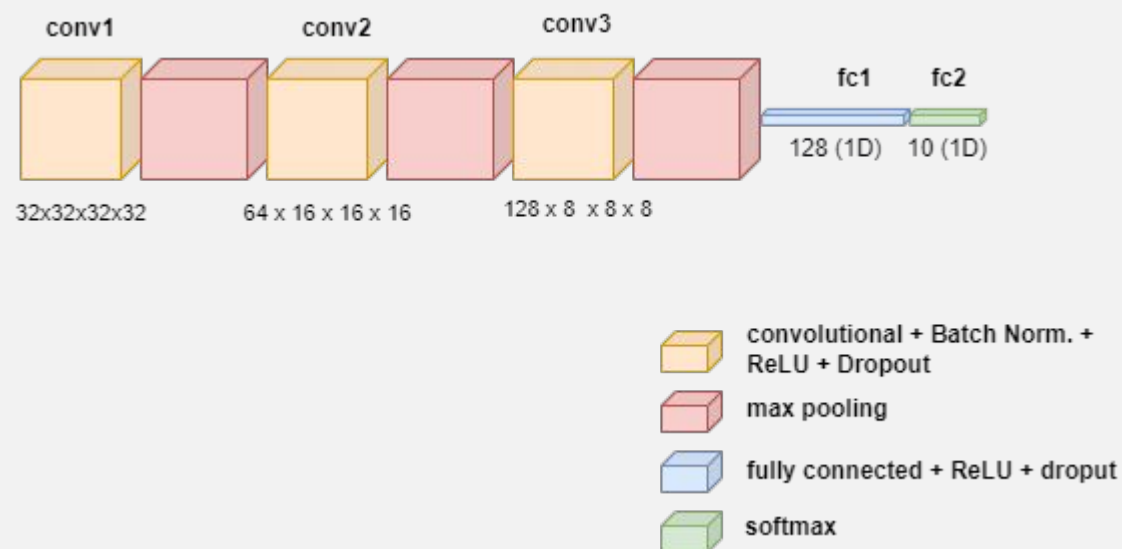
Reference: [Research Gate](#)

MODEL



# THE WEAK MODELS STRUCTURE

Layer (type)	Output Shape	Param #
Conv3d-1	[-1, 32, 32, 32, 32]	896
BatchNorm3d-2	[-1, 32, 32, 32, 32]	64
ReLU-3	[-1, 32, 32, 32, 32]	0
Dropout-4	[-1, 32, 32, 32, 32]	0
MaxPool3d-5	[-1, 32, 16, 16, 16]	0
Conv3d-6	[-1, 64, 16, 16, 16]	55,360
BatchNorm3d-7	[-1, 64, 16, 16, 16]	128
ReLU-8	[-1, 64, 16, 16, 16]	0
Dropout-9	[-1, 64, 16, 16, 16]	0
MaxPool3d-10	[-1, 64, 8, 8, 8]	0
Conv3d-11	[-1, 128, 8, 8, 8]	221,312
BatchNorm3d-12	[-1, 128, 8, 8, 8]	256
ReLU-13	[-1, 128, 8, 8, 8]	0
Dropout-14	[-1, 128, 8, 8, 8]	0
MaxPool3d-15	[-1, 128, 4, 4, 4]	0
Linear-16	[-1, 128]	1,048,704
ReLU-17	[-1, 128]	0
Dropout-18	[-1, 128]	0
Linear-19	[-1, 10]	1,290



!! Represent in 3D for model flow, but 3DConv layer has 4 dimension

# TRAINING & EXPERIMENTAL RESULTS





# TRAINING WEAK MODELS

Epochs per model:100

- Converge quickly <50 Epochs

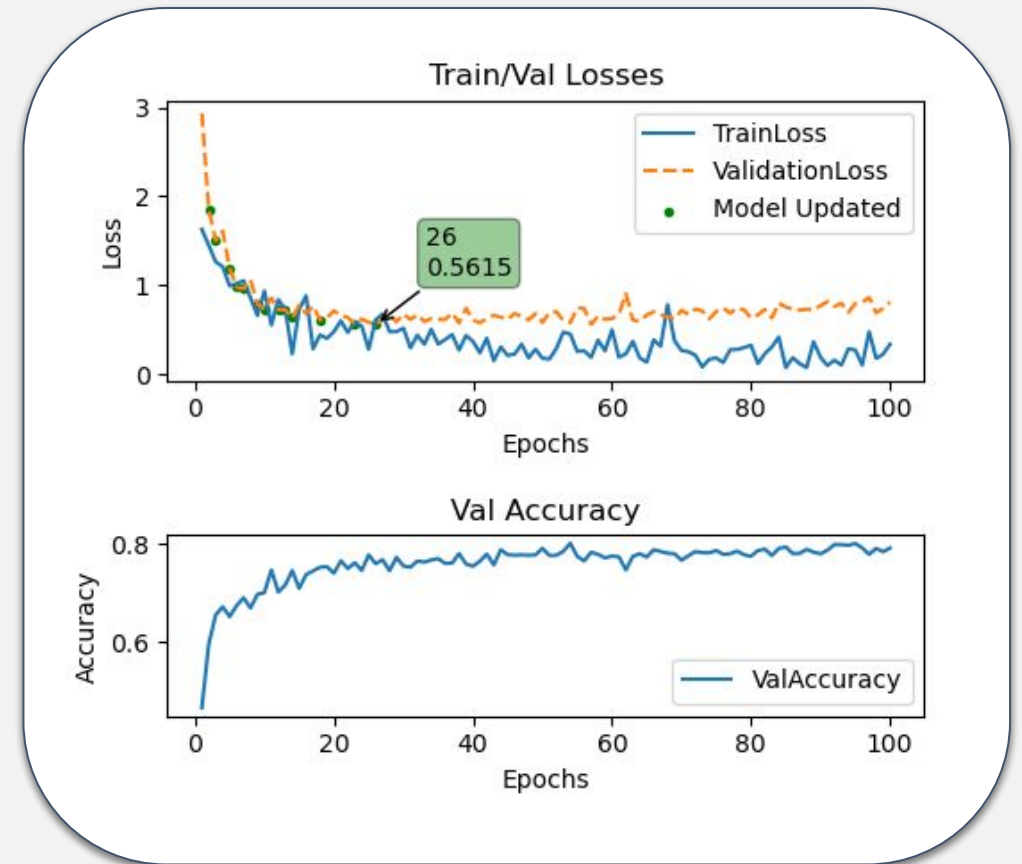
Learning rate:  $1e-3$

Optimizer: ADAM

Loss: Cross Entropy

Max Pooling Kernel: 2x2x2

Max Pooling Stride: 2x2x2



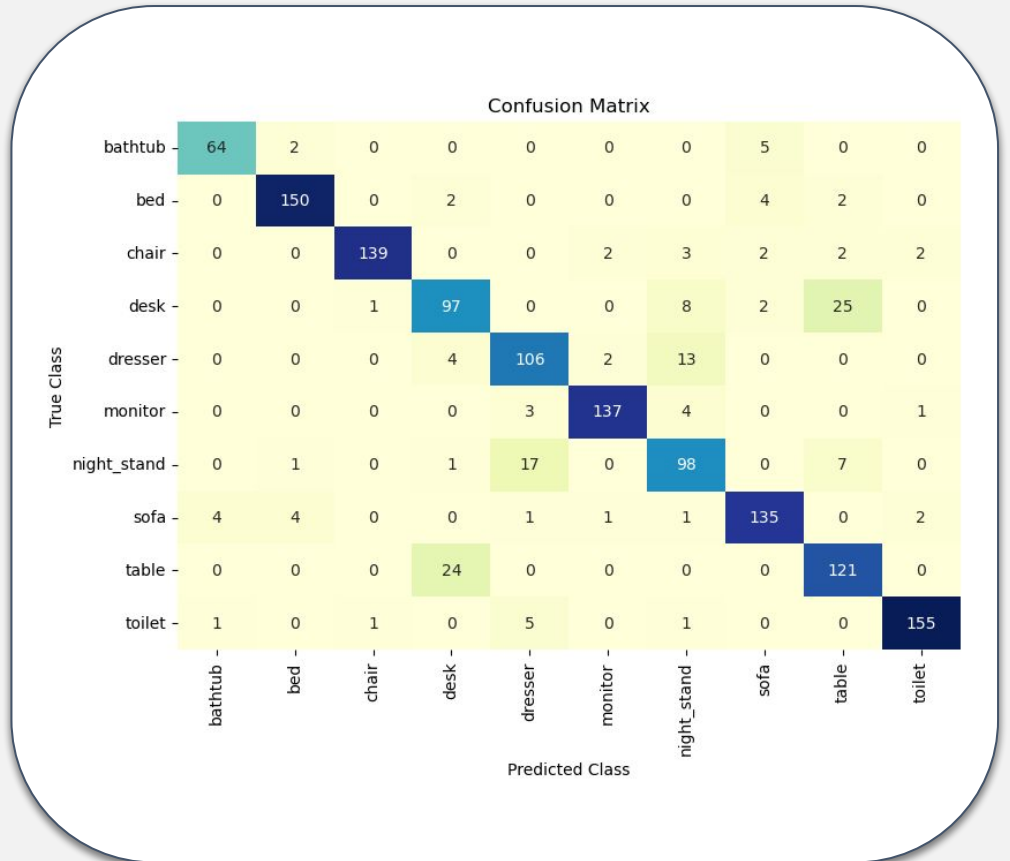
## RESULTS ON VALIDATION SET

CLASS		M0	M1	M2	M3	M4	EN
0	bathtub	0.98	0.98	0.94	0.97	0.93	0.98
1	bed	0.92	0.90	0.92	0.96	0.94	0.94
2	chair	0.96	0.91	0.98	0.94	0.96	0.97
3	desk	0.62	0.67	<b>0.80</b>	0.75	0.65	0.78
4	dresser	0.74	0.73	0.71	0.83	0.74	0.77
5	monitor	0.94	0.97	0.97	0.96	0.97	0.96
6	night_stand	0.71	0.75	0.75	0.81	0.67	0.75
7	sofa	0.88	0.76	0.81	0.87	0.90	0.88
8	table	0.67	0.77	0.81	0.87	0.90	0.88
9	toilet	<b>0.99</b>	0.90	0.95	0.91	0.98	0.97

# RESULTS ON TEST SET

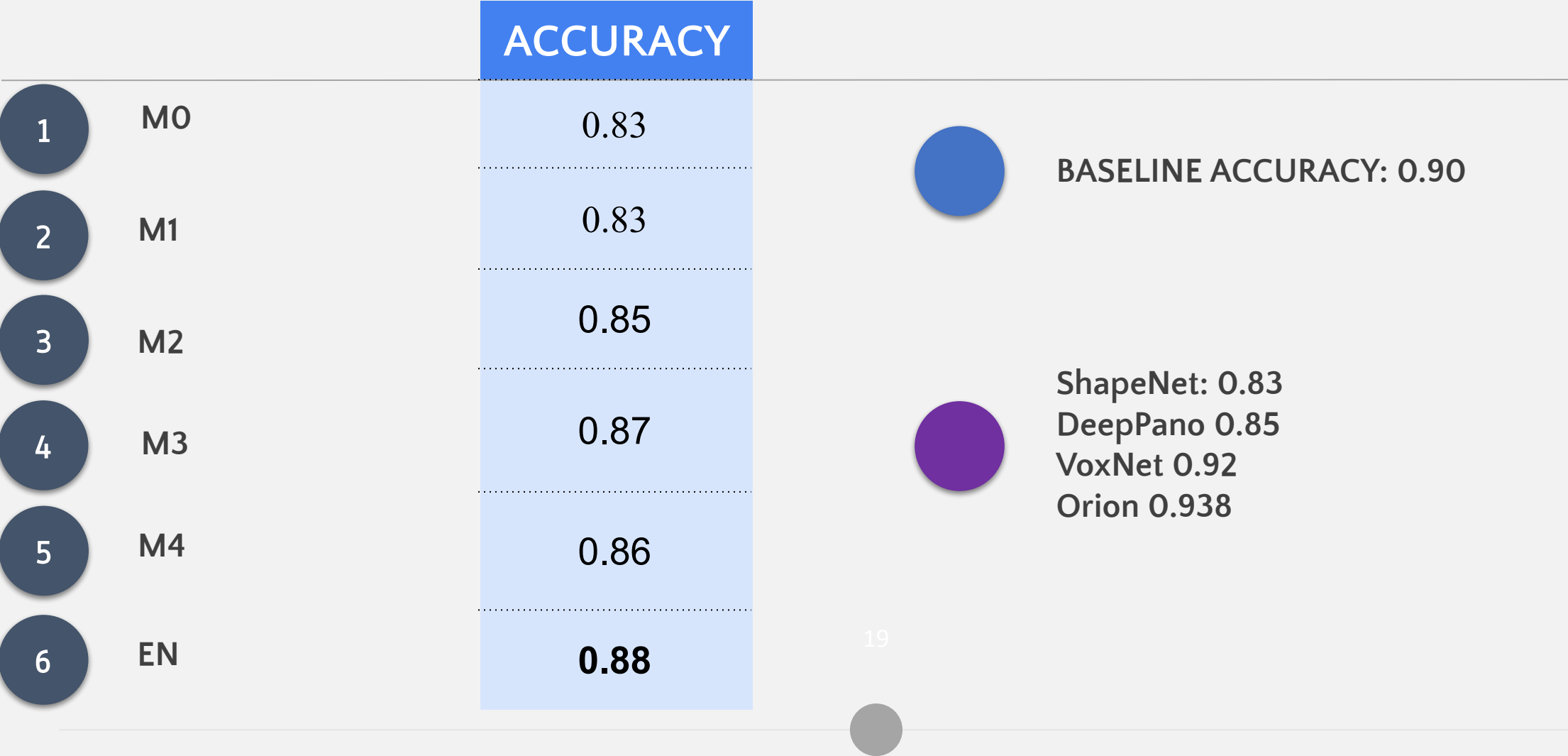
## Ensembled model results:

- Good diagonal Confusion matrix
- Errors are “Symmetric”



*Confusion matrix of the ensemble model*

# RESULTS ON TEST SET



# CONCLUSIONS AND FUTURE WORK



# CONCLUSIONS



Model has good accuracy



Dynamic, can be adapted on the computer in which is run

## FUTURE WORK



Adding more data and Models



Implements a way to capture long-range context information

**END THANK YOU**





# NOTEBOOK DEMO



NNDL\_Project\_BAGGING\_kag... Draft saved

File Edit View Run Add-ons Help

Share Save Version 9

+ + - ✂ 📄 ▶ ⏮ Run All Markdown Draft Session off (run a cell to start)

Parameters/Variables for the Training Session

+ Code + Markdown

[1]:

```
# Model need to be trained
MODEL = "VGNet_v3" #@param ["VGNetOrion_extended", "VGNet_v3"]{type:"string"}
OPTIMIZER = "Adam" #@param ["Adam", "SGD"]{type:"string"}
LEARNING_RATE = 1e-3 #@param ["1e-2", "1e-3", "1e-4"]{type:"raw"}
##@markdown <pre>Normally LR would be:
##@markdown <b>0.001</b> for ADAM
##@markdown and <b>0.0001</b> with SGD</pre>

# Put to false if you need to do only the evaluation
TRAIN = False
BAGGING = True # for bagging (TODO: SEPARATE EPOCHS & old)
MODELS_TO_TRAIN = [0,1,2,3,4] # max count 2 (only 2 gpus available) could be

# If FALSE: 1) disables tqdm LOGGING
#           2) Re-Fetches model from GDrive
#           3) Saved plot as image
DRAFT_MODE = False # True = running in background

# To resume the training set the following, remember to set also the correct
epochs= 100 #@param {type:"number"} #100epochs require ~3h for training

## TODO: CHANGE also the reloading code
old = 100 #@param {type:"number"}
olds = [100, 100, 100, 100, 100]

# 2) Fetching model id of GDrive
MODEL_FILE_GDRIVE_ID = "1wb1FUQIqWlIXxwHBXsX95JuRnMTGgDIX"
BAGGING_MODELS_FILES_GDRIVE_IDS = [
    "1IdeoHu9UxsL1IvidkdpqkqWnp_SxZD00", #m0
    "1wu15ngwwkJAKlWYAXz1ZQD3BNW7MF-K", #m1
    "1Rj3sW44kwyQeBD9KzoYVCX3m_MxnM9Ynv", #m2
    "1UG6qkzXZyTdk8S81ThzdaYRQ53d87s1P", #m3
    "1NZVKPfZ0xcshnPUEphYhQmQXkGTocCac", #m4
]
best_val_loss = 10#1.6286746263504028 #@param {type:"number"}
#best_val_loss_epoc = 662 #@param {type:"number"}

PRINT = False

# if you want to save models every 100 epochs
SNAPSHOT = True #@param ["True", "False"] {type:"raw"}
# if you want to save the model on last epoch KEEP IN MIND: vgnet_v3_adam.1e
RESAVE_LAST_SNAPSHOT = True #@param ["True", "False"] {type:"raw"}

#params = [{"MODEL":MODEL}, {"OPTIMIZER":OPTIMIZER}], LEARNING_RATE, epochs,
#params = [MODEL, OPTIMIZER, LEARNING_RATE, epochs, old, best_val_loss, best_val_loss_epoc]
print("STARTING WITH params")
# for param in params:
#     param_name = [k for k, v in list(locals().items()) if v == param][0]
#     print(f" {param_name}:{(16-len(param_name))*' '} {param}")

print(f" MODEL:{(16-len('MODEL'))*' '} {MODEL}")
print(f" OPTIMIZER:{(16-len('OPTIMIZER'))*' '} {OPTIMIZER}")
print(f" LEARNING_RATE:{(16-len('LEARNING_RATE'))*' '} {LEARNING_RATE}")
print(f" TRAIN:{(16-len('TRAIN'))*' '} {TRAIN})"
```