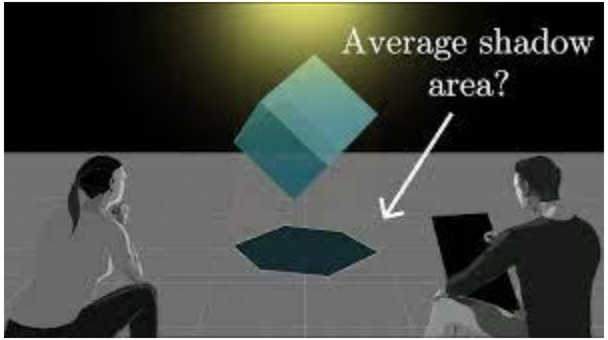
MLHO
3D object recognition from 2D Projection

Mohamad Al Farhan
Emil Reiter
Mahmoud Sharaf

k

Inspiration and Motivation

3Blue1brown: average area of a shadow



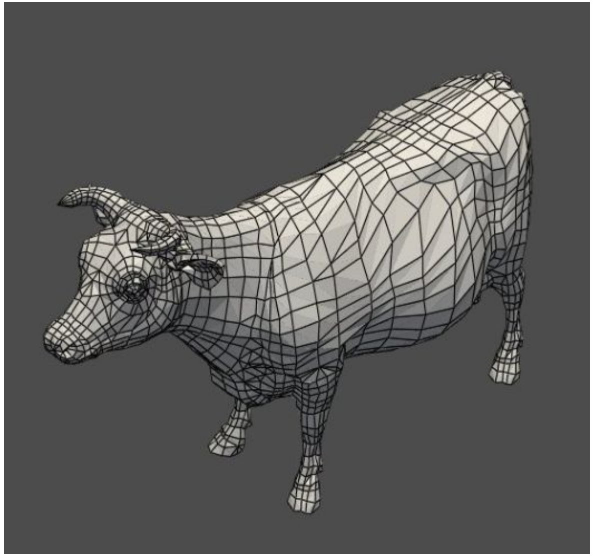
Source: https://www.youtube.com/watch?v=ltLUadnCyi0

Question: Can we identify the original object from its shadow?

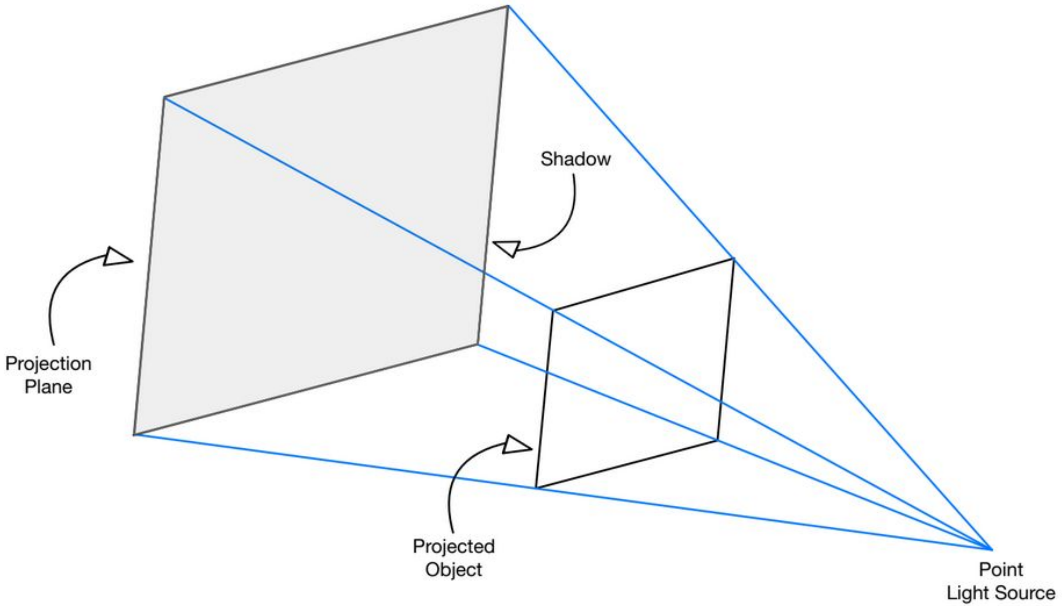
Data generation and preprocessing

3D Shapes

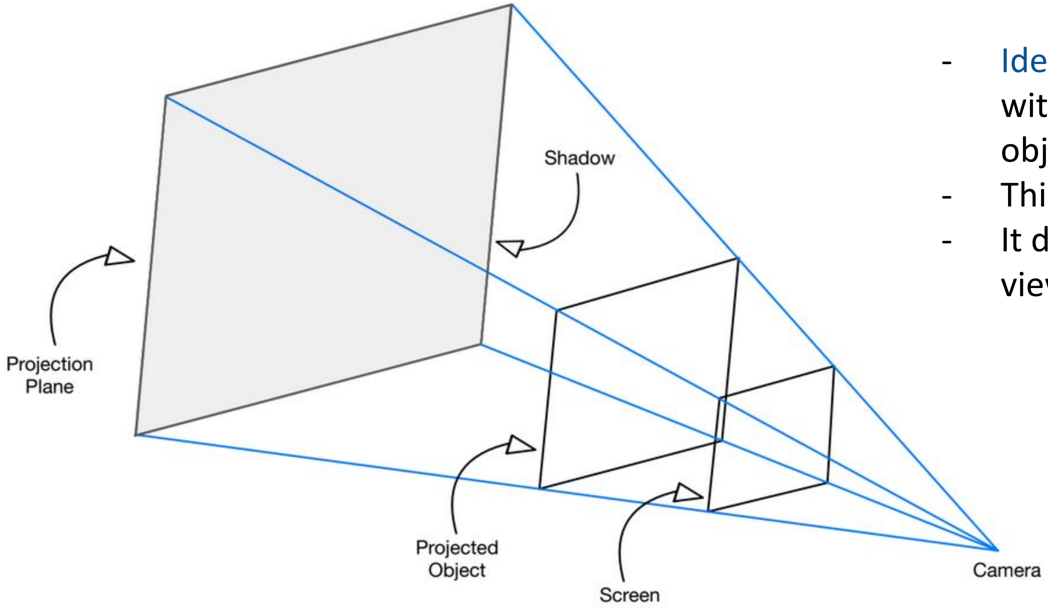
- Objects are constructed with polygons
- We have a 3D mesh for each object
- Objects include:
    - Cube
    - Pyramid
    - Tetrahedron
    - Diamond
    - Dodecahedron
    - Icosahedron
    - Torus
    - Cylinder
    - Cow
    - Human



Perspective Projection



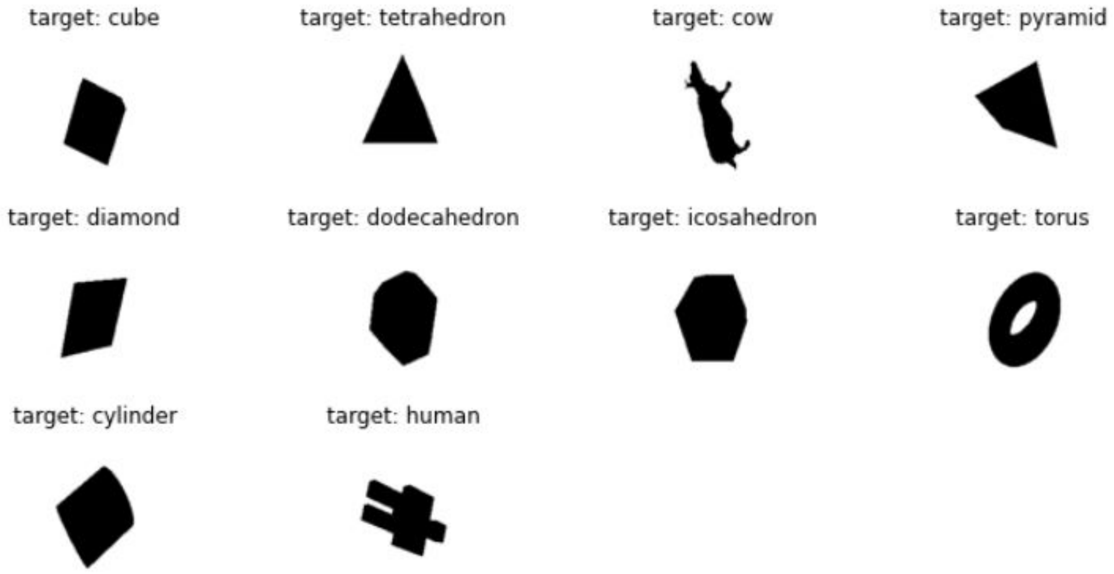
Perspective Projection



- Idea: replace the light source with a viewer and project the object to the screen instead.
- This inverses the parity
- It doesn’t matter because we view all orientations anyway

Perspective Projection

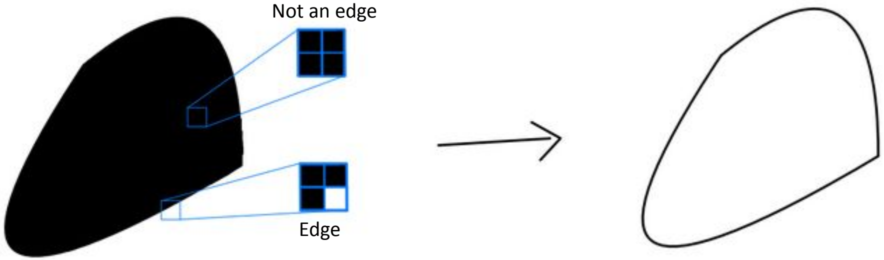
Results: We have shadows like this with 1300 samples per shape



Preprocessing

Primitive edge detection:

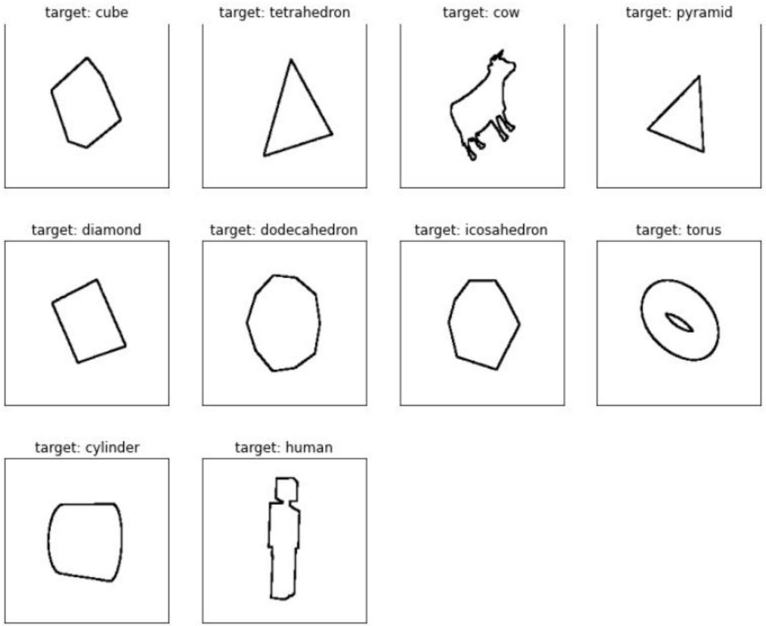
- Loop over all pixels in the image
- If a pixel has a different value than its neighbours consider it part of the edge



Preprocessing

Primitive edge detection:

Results

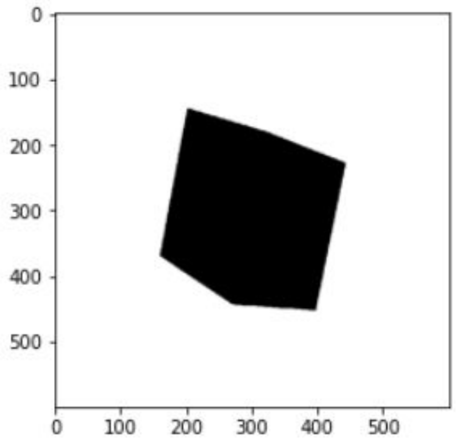


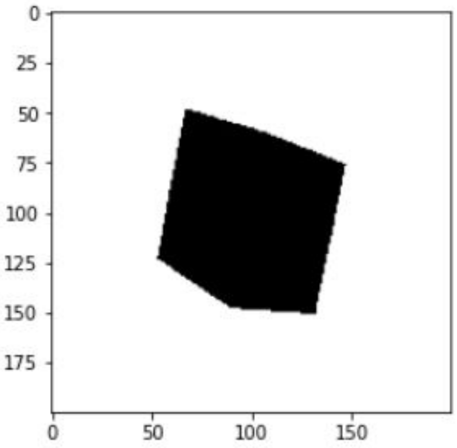
Preprocessing

Data compression:

- Cast pixel values as booleans
- Change (and reduce) the image resolution

-due to memory issues, all images converted to 200x200, dtype='bool'

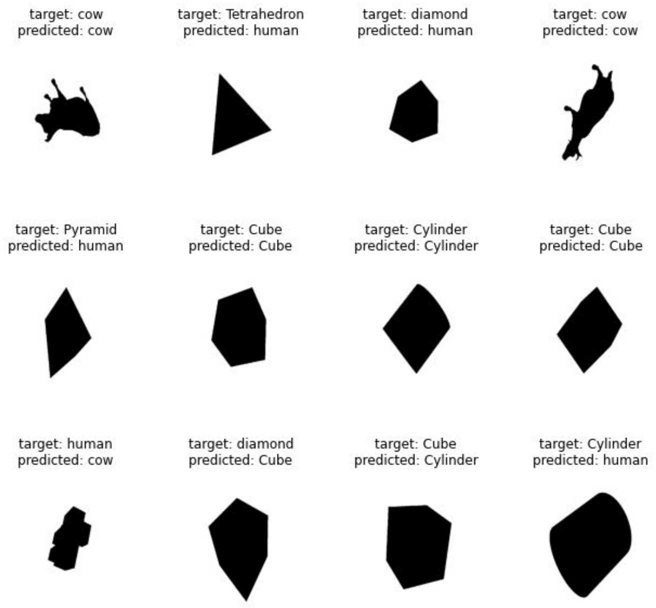




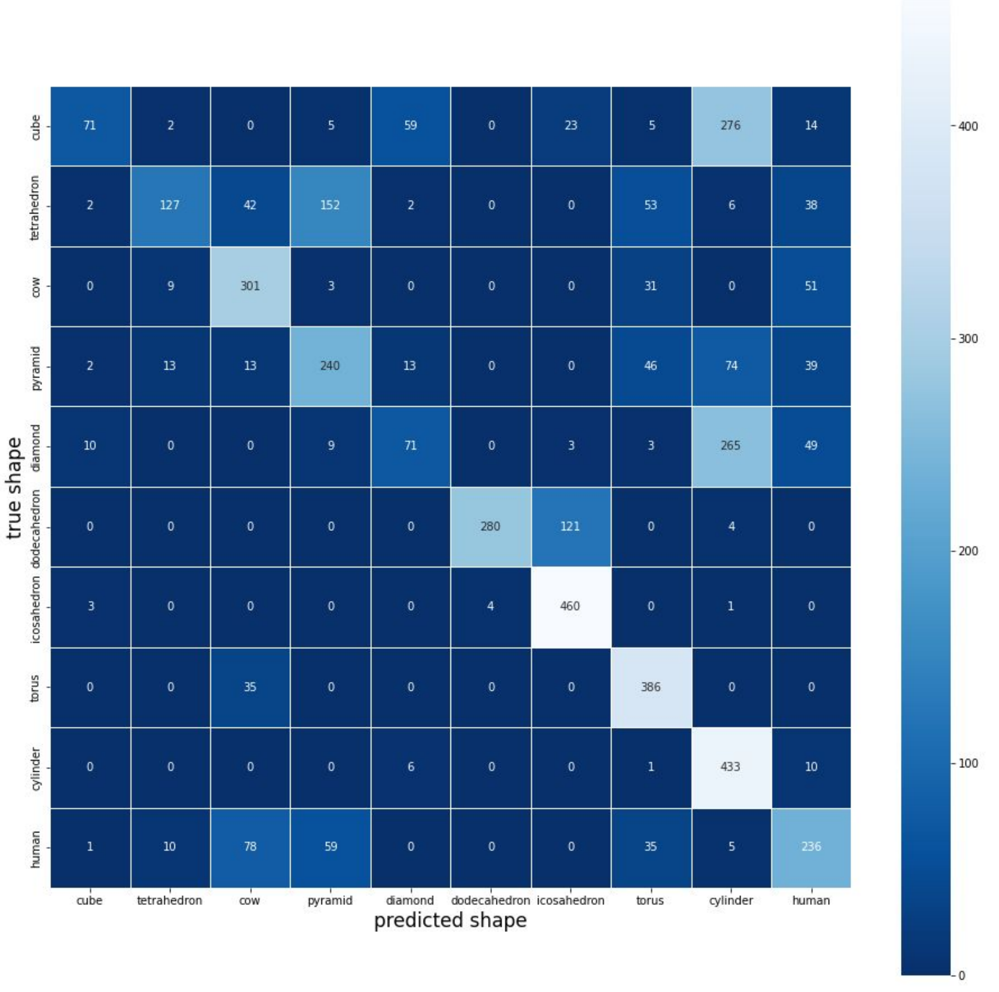
Classification

Multinomial Logistic Regression

- Trained with L2 regularization
- Accuracy score ≈ 0.60



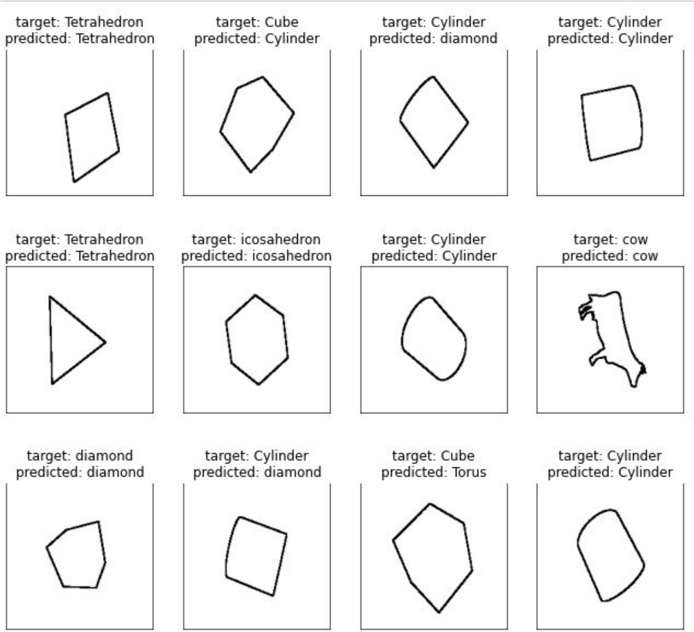
Multinomial Logistic Regression



Multinomial Logistic Regression

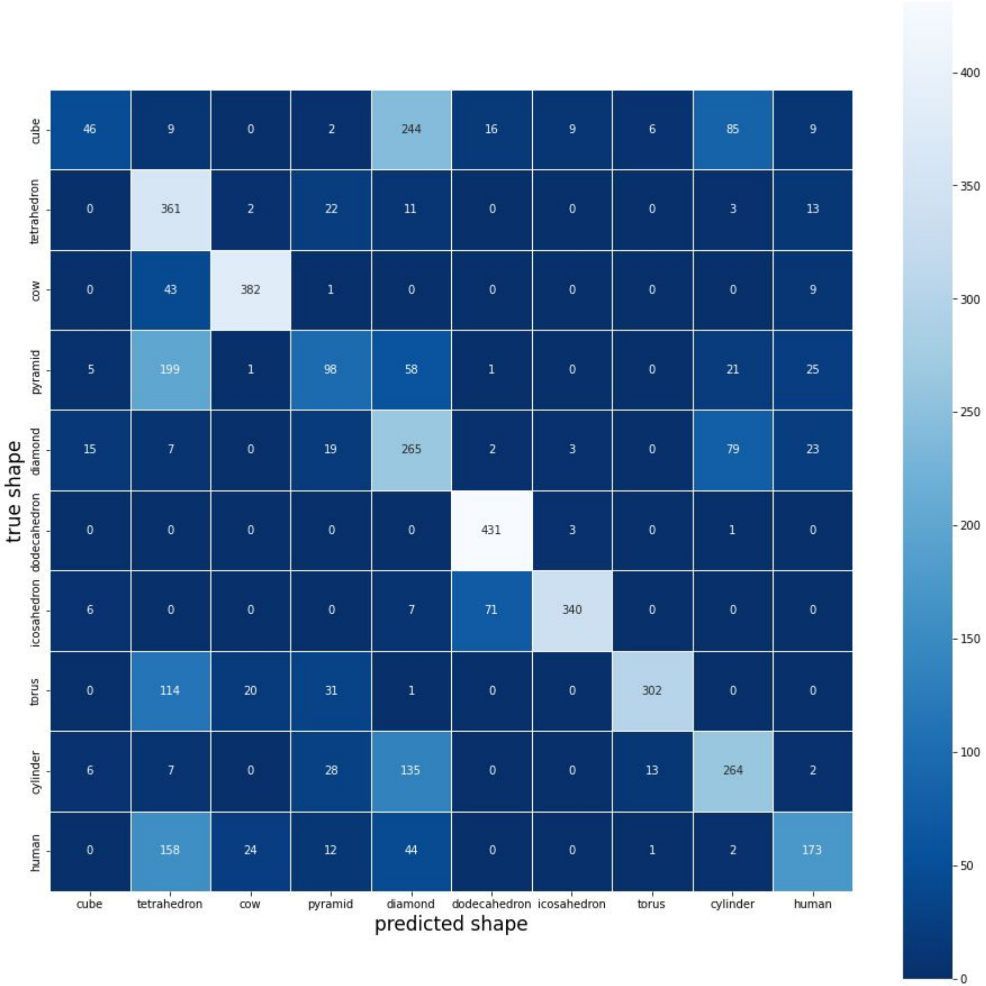
Slight improvement with the edges dataset

Score ≈0.62



Multinomial Logistic Regression

Edges Dataset



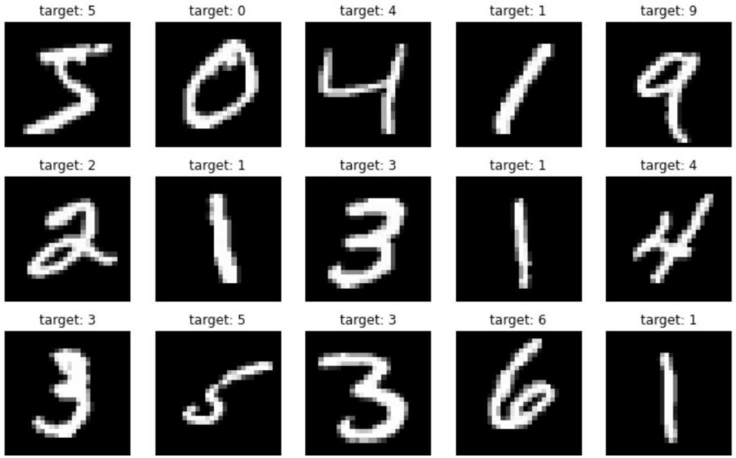
Deep Neural Network

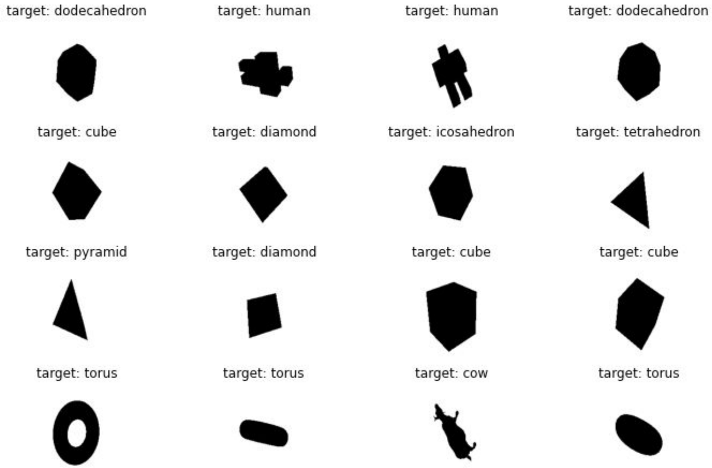
-used Kaggle to train neural networks

-CNNs are default neural networks for image classification, but DNN could also be used

-in the class we did MNIST with DNN, we used 2 models that had 95% and 98% accuracy, but our dataset is more complex

MNIST vs Dataset





Models from the class

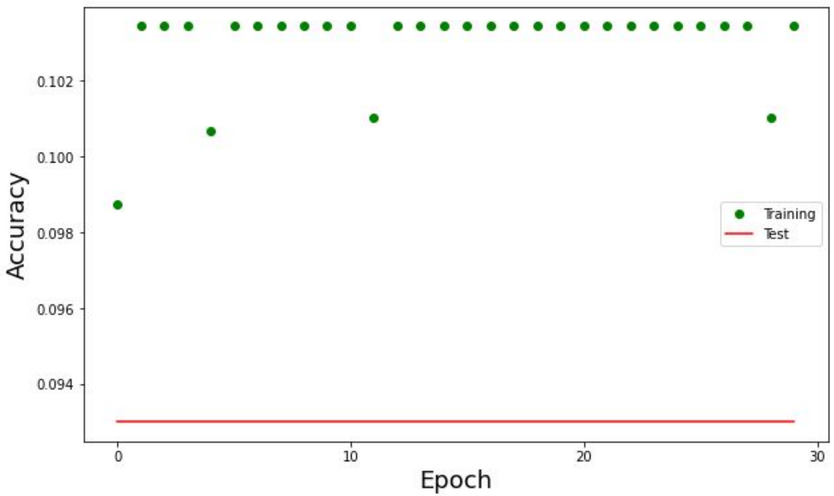
Layer (type)	Output Shape	Param #
dense_28 (Dense)	(None, 15)	600015
dense_29 (Dense)	(None, 10)	160

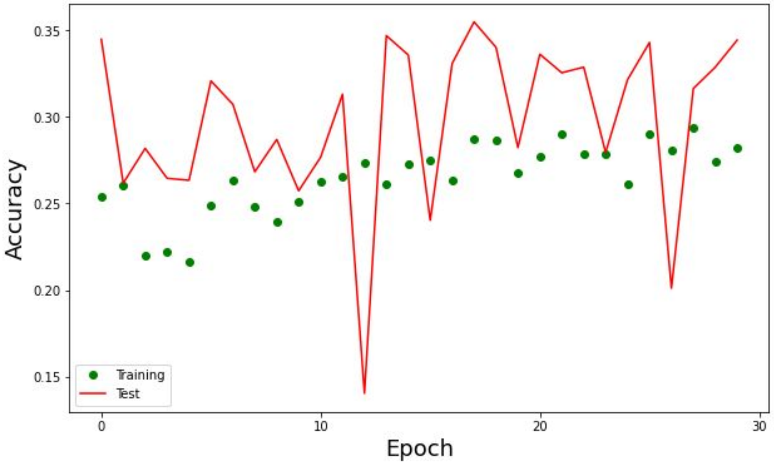
Total params: 600,175
Trainable params: 600,175
Non-trainable params: 0

Layer (type)	Output Shape	Param #
dense_23 (Dense)	(None, 512)	20480512
dropout_4 (Dropout)	(None, 512)	0
dense_24 (Dense)	(None, 512)	262656
dropout_5 (Dropout)	(None, 512)	0
dense_25 (Dense)	(None, 10)	5130

Total params: 20,748,298
Trainable params: 20,748,298
Non-trainable params: 0

Results with models from class





Model that worked the best

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_18 (Dense)             (None, 200)               72000200
_________________________________________________________________
dense_19 (Dense)             (None, 100)               20100
_________________________________________________________________
dropout_7 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_20 (Dense)             (None, 50)                5050
_________________________________________________________________
dense_21 (Dense)             (None, 4)                 204
=================================================================
Total params: 72,025,554
Trainable params: 72,025,554
Non-trainable params: 0
_________________________________________________________________
```

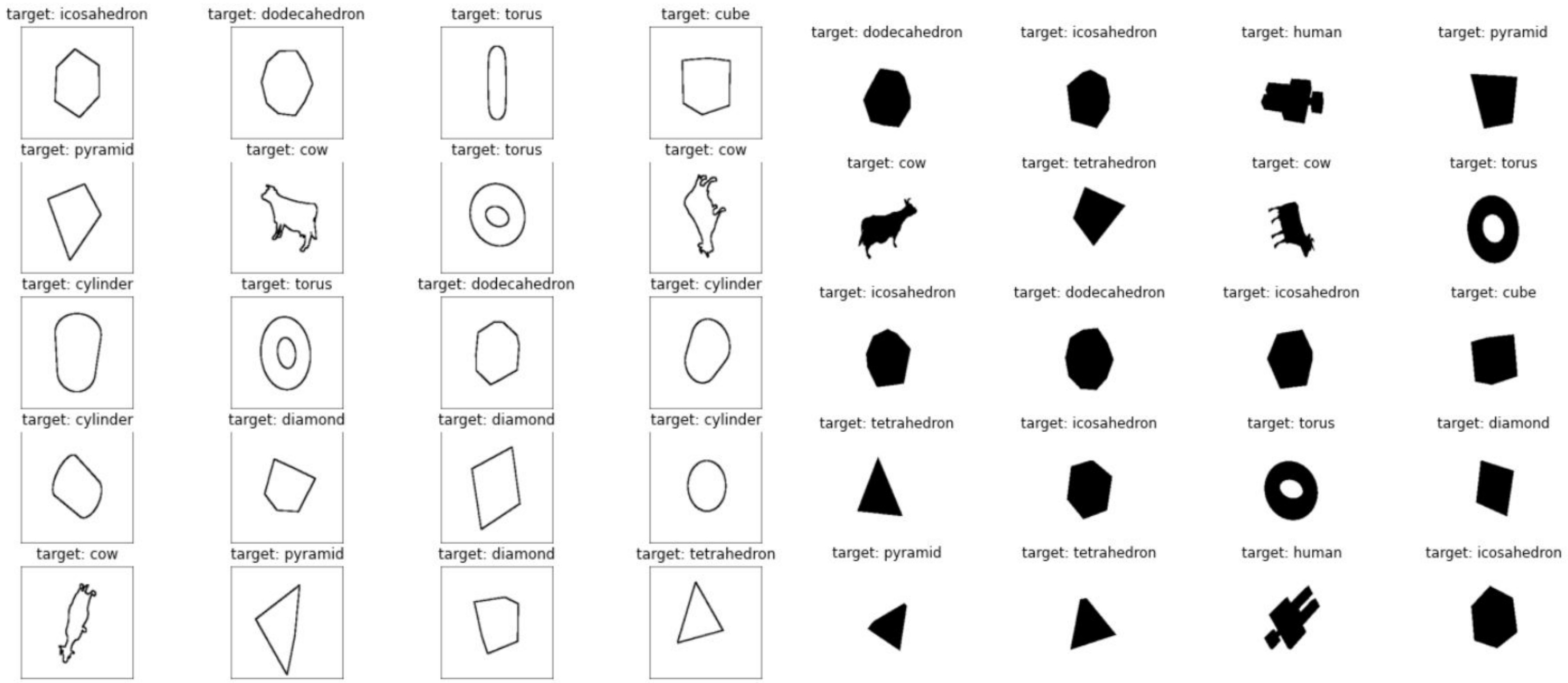
-with dropout=0.2 and learning rate=0.00001, optimizer Adam, batch size = 40

10 shapes

-All 10 shapes: cube, tetrahedron, cow, pyramid, diamond, dodecahedron, icosahedron, torus, cylinder, human

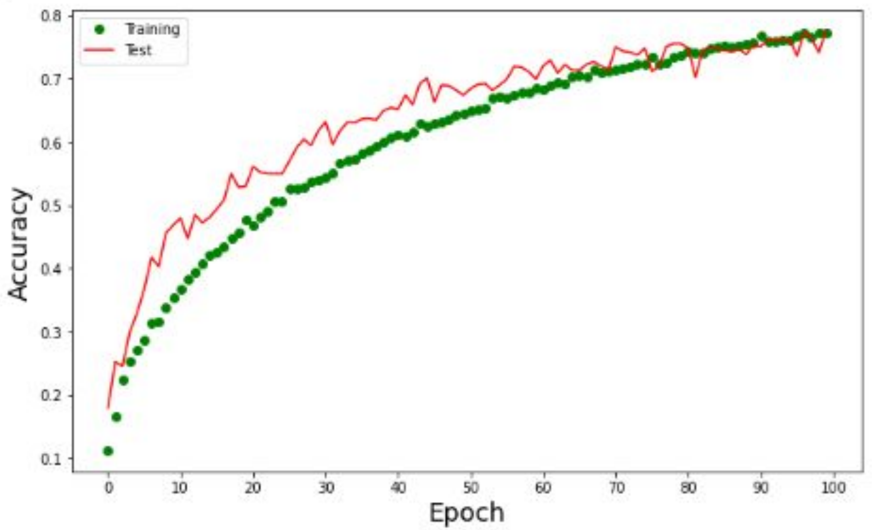
-1300 samples per shape

Edges and full shapes

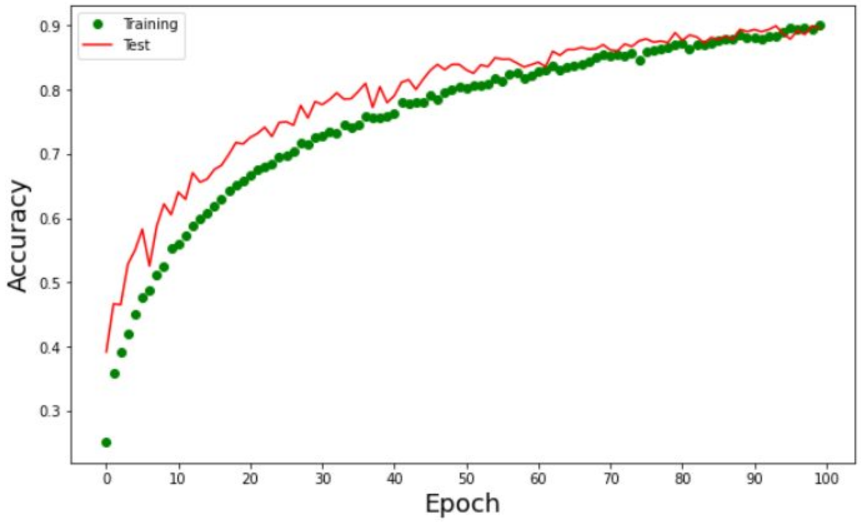


Results

Edges



Full shapes



Classification reports

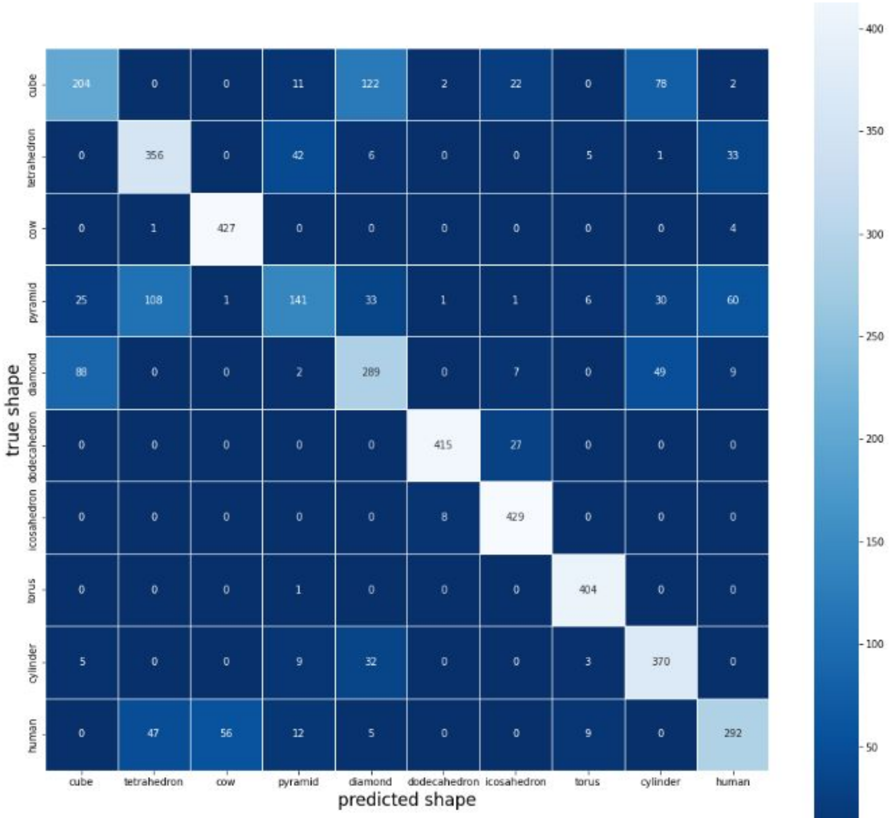
Edges

	precision	recall	f1-score	support
cube	0.63	0.46	0.53	441
tetrahedron	0.70	0.80	0.75	443
cow	0.88	0.99	0.93	432
pyramid	0.65	0.35	0.45	406
diamond	0.59	0.65	0.62	444
dodecahedron	0.97	0.94	0.96	442
icosahedron	0.88	0.98	0.93	437
torus	0.95	1.00	0.97	405
cylinder	0.70	0.88	0.78	419
human	0.73	0.69	0.71	421
accuracy			0.78	4290
macro avg	0.77	0.77	0.76	4290
weighted avg	0.77	0.78	0.76	4290

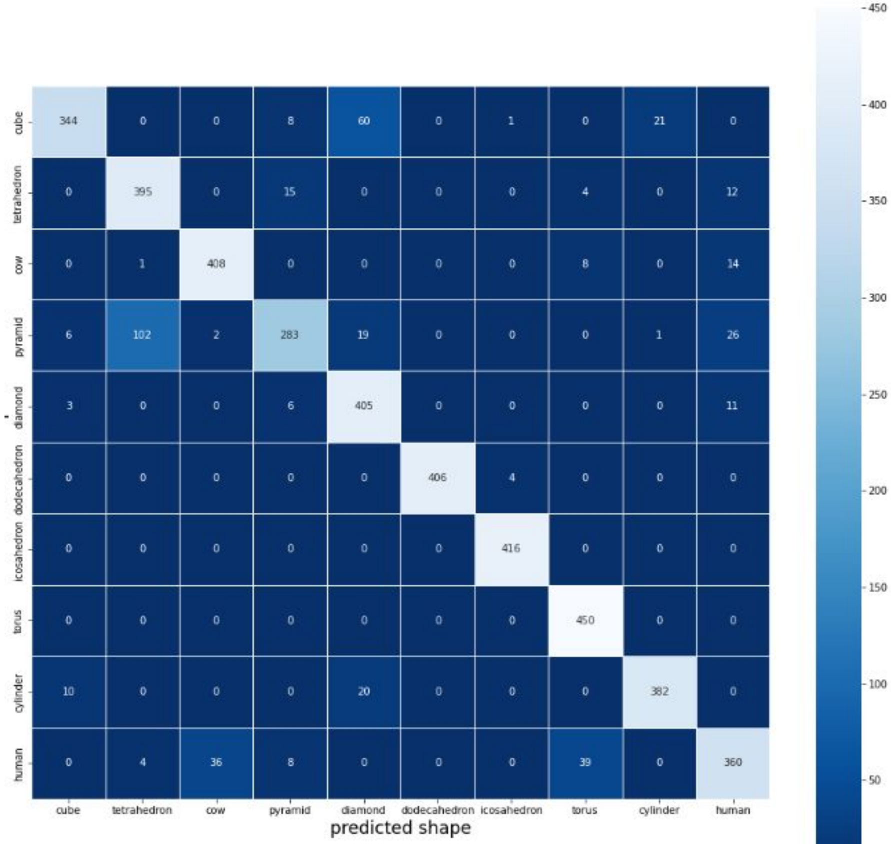
Full shapes

	precision	recall	f1-score	support
cube	0.95	0.79	0.86	434
tetrahedron	0.79	0.93	0.85	426
cow	0.91	0.95	0.93	431
pyramid	0.88	0.64	0.75	439
diamond	0.80	0.95	0.87	425
dodecahedron	1.00	0.99	1.00	410
icosahedron	0.99	1.00	0.99	416
torus	0.90	1.00	0.95	450
cylinder	0.95	0.93	0.94	412
human	0.85	0.81	0.83	447
accuracy			0.90	4290
macro avg	0.90	0.90	0.90	4290
weighted avg	0.90	0.90	0.89	4290

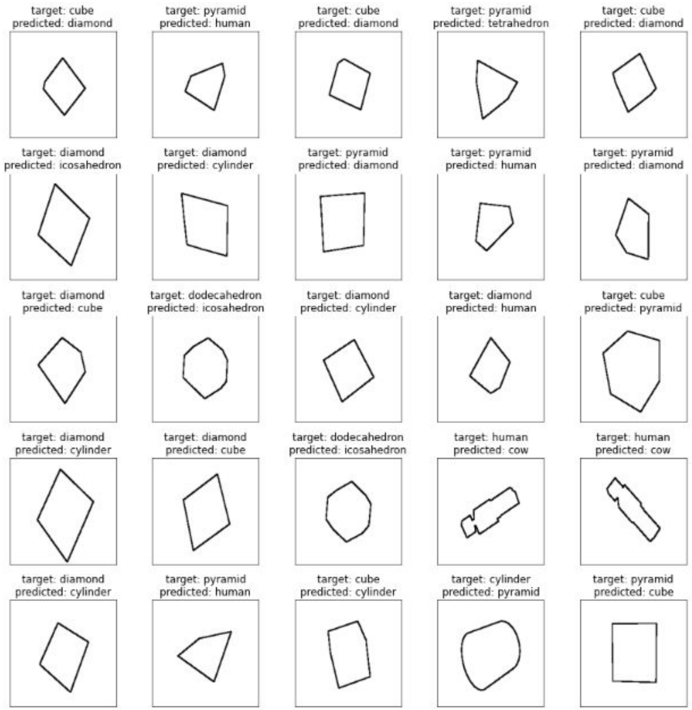
Edges

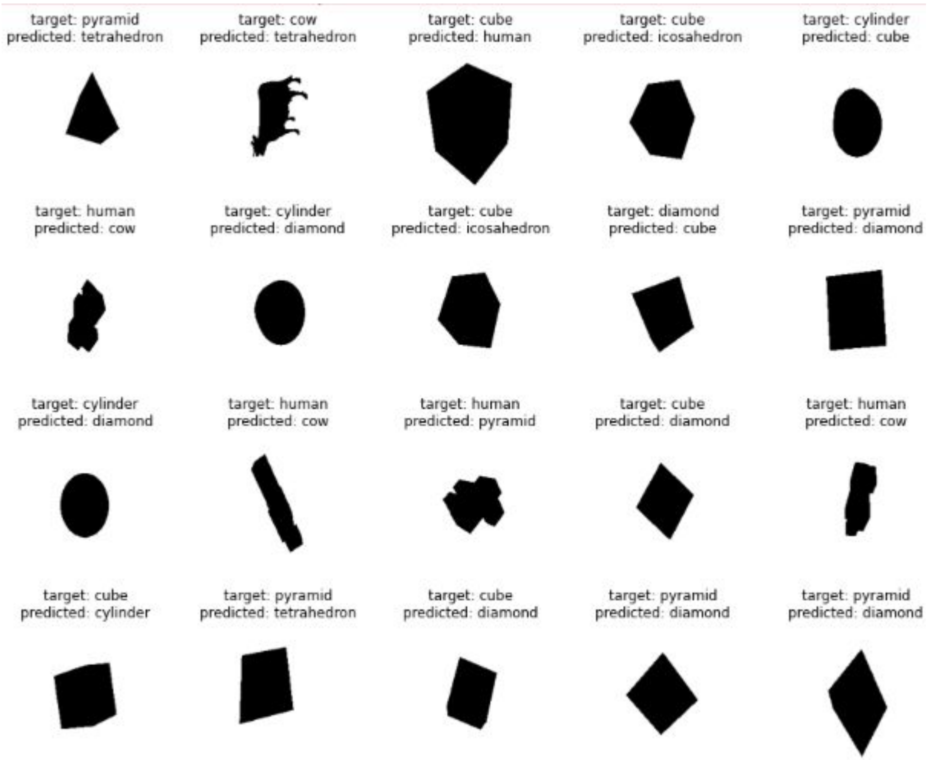


Full shapes

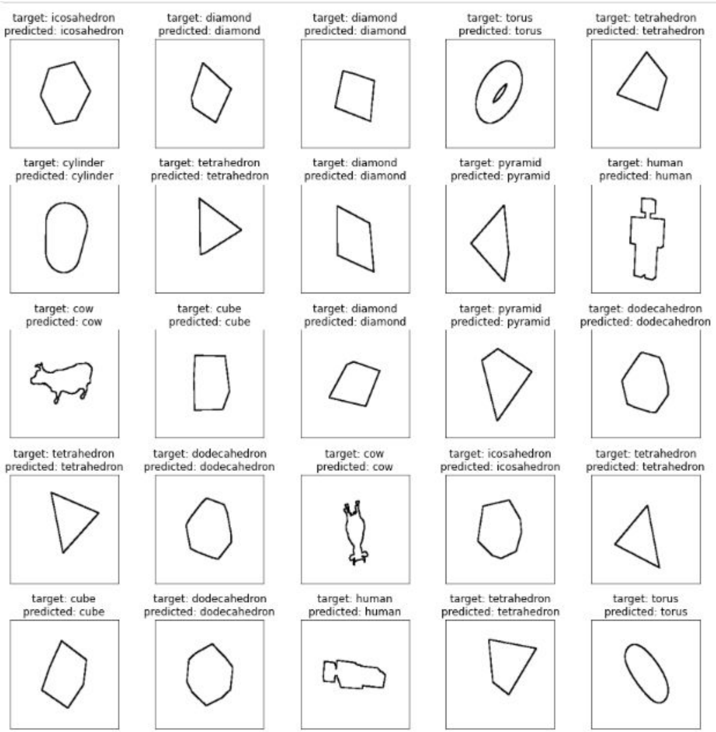


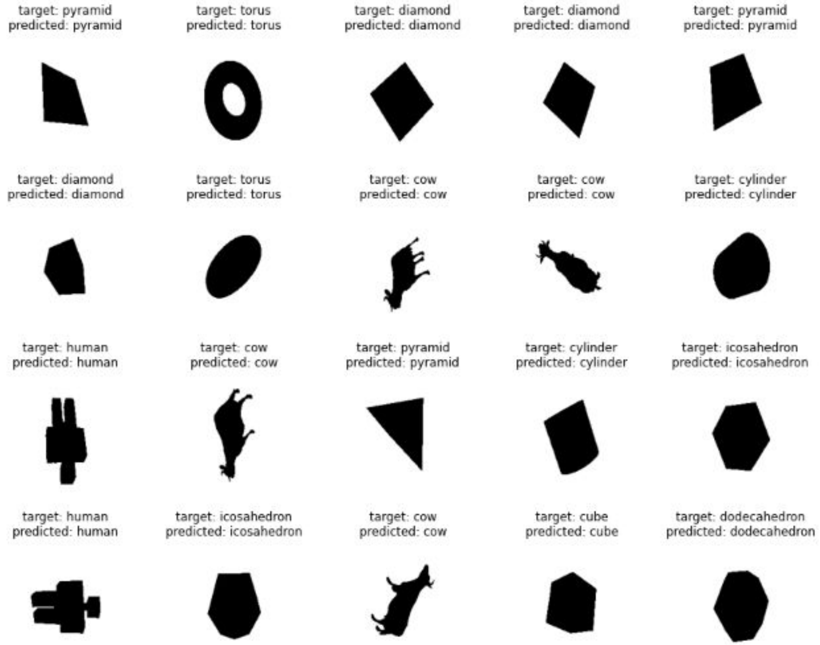
Wrong predictions





Correct predictions



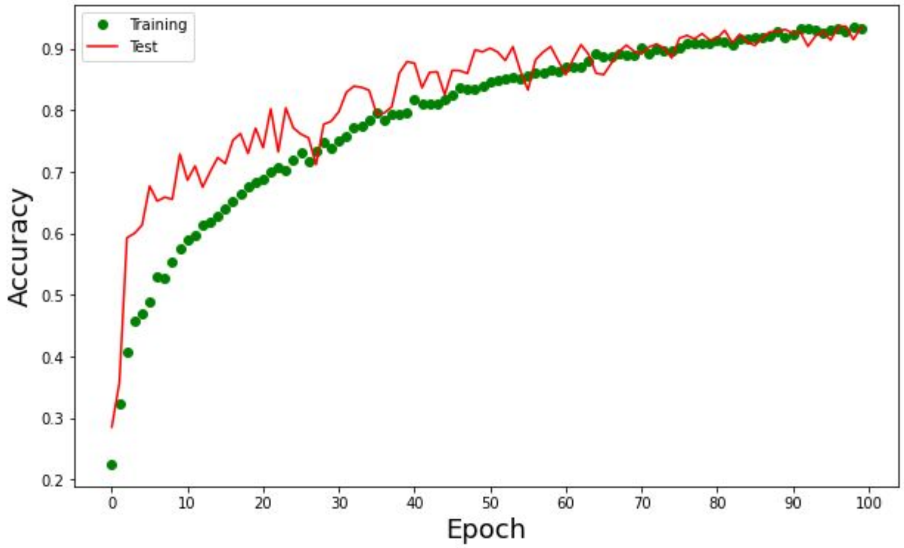


-7 shapes: cube, cow, dodecahedron, icosahedron, torus, cylinder, human

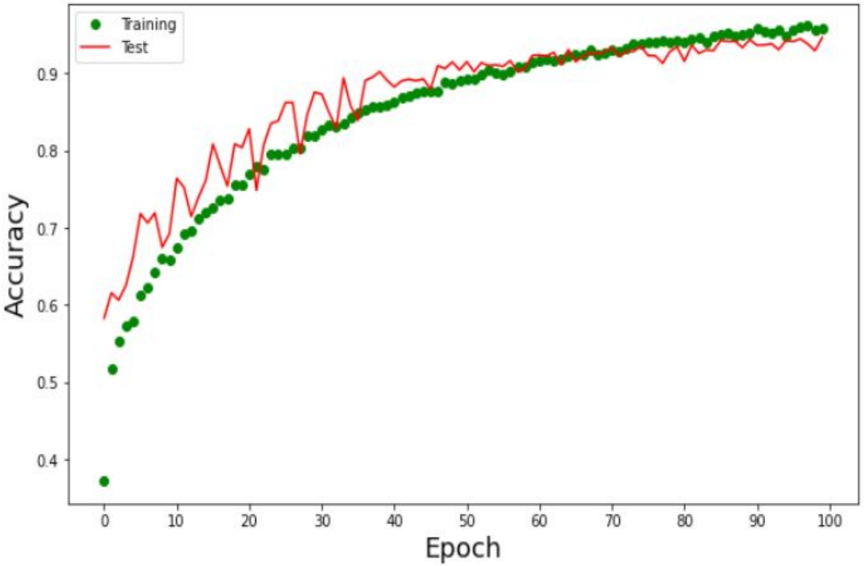
-Less ambiguity than for 10 shapes: cube, diamond, tetrahedron and pyramid are problematic

-Results:

Edges



Full shapes



Classification reports

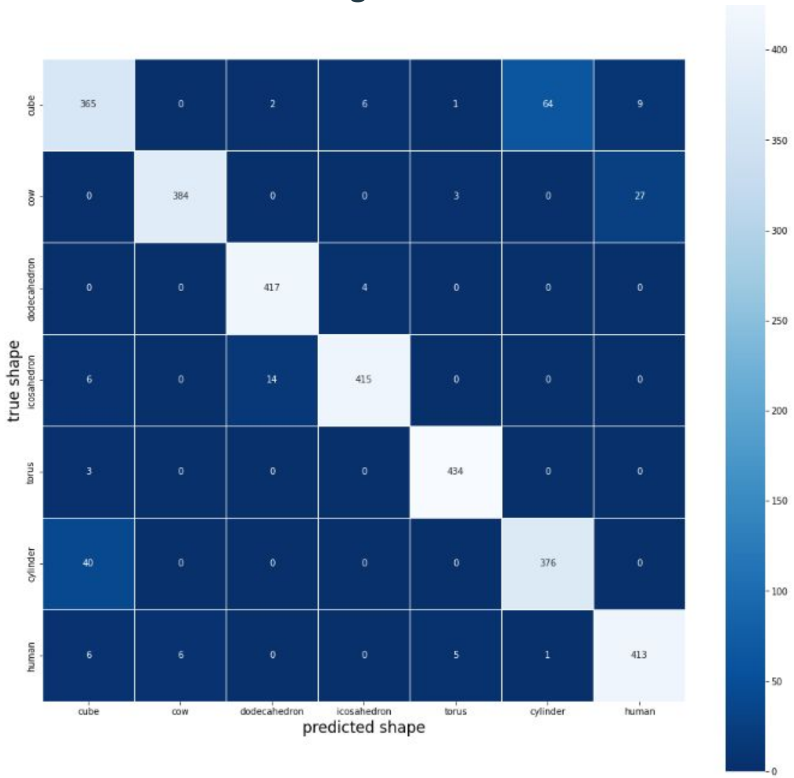
Edges

	precision	recall	f1-score	support
cube	0.87	0.82	0.84	447
cow	0.98	0.93	0.96	414
dodecahedron	0.96	0.99	0.98	421
icosahedron	0.98	0.95	0.97	435
torus	0.98	0.99	0.99	437
cylinder	0.85	0.90	0.88	416
human	0.92	0.96	0.94	431
accuracy			0.93	3001
macro avg	0.94	0.93	0.93	3001
weighted avg	0.93	0.93	0.93	3001

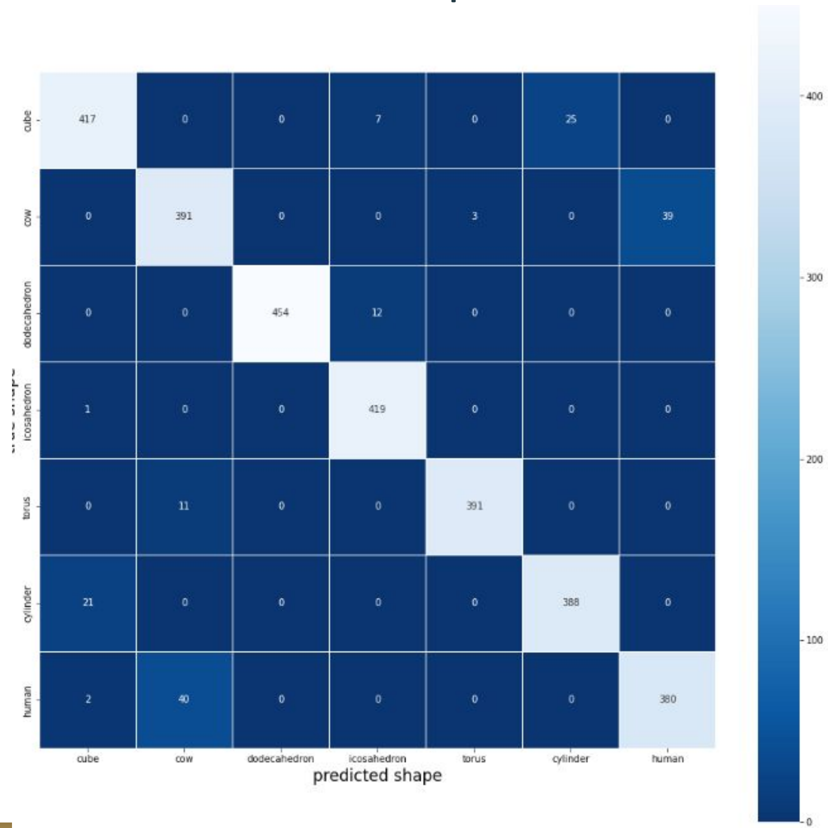
Full shapes

	precision	recall	f1-score	support
cube	0.95	0.93	0.94	449
cow	0.88	0.90	0.89	433
dodecahedron	1.00	0.97	0.99	466
icosahedron	0.96	1.00	0.98	420
torus	0.99	0.97	0.98	402
cylinder	0.94	0.95	0.94	409
human	0.91	0.90	0.90	422
accuracy			0.95	3001
macro avg	0.95	0.95	0.95	3001
weighted avg	0.95	0.95	0.95	3001

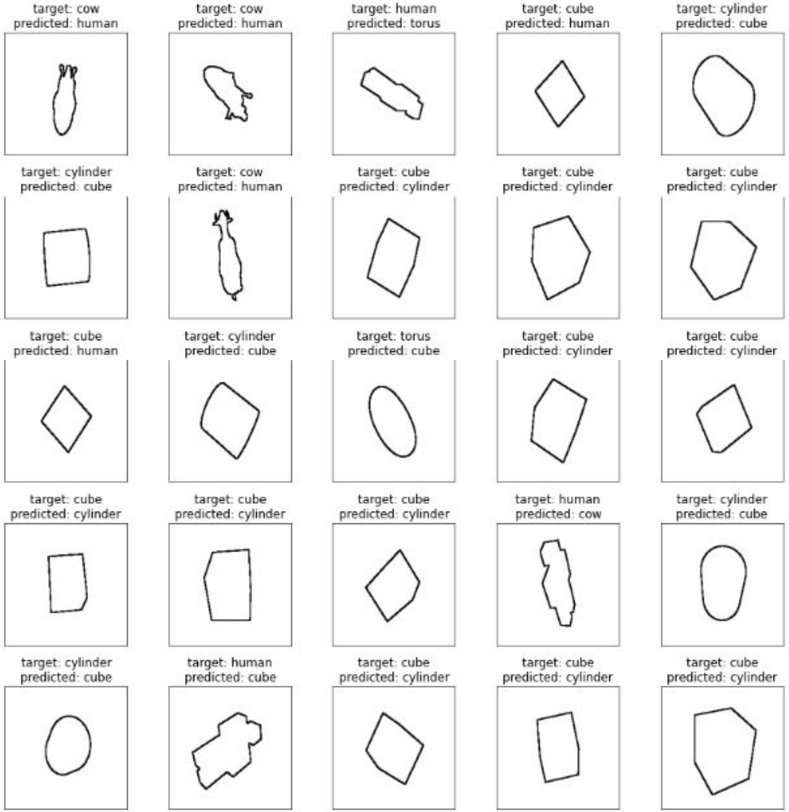
Edges

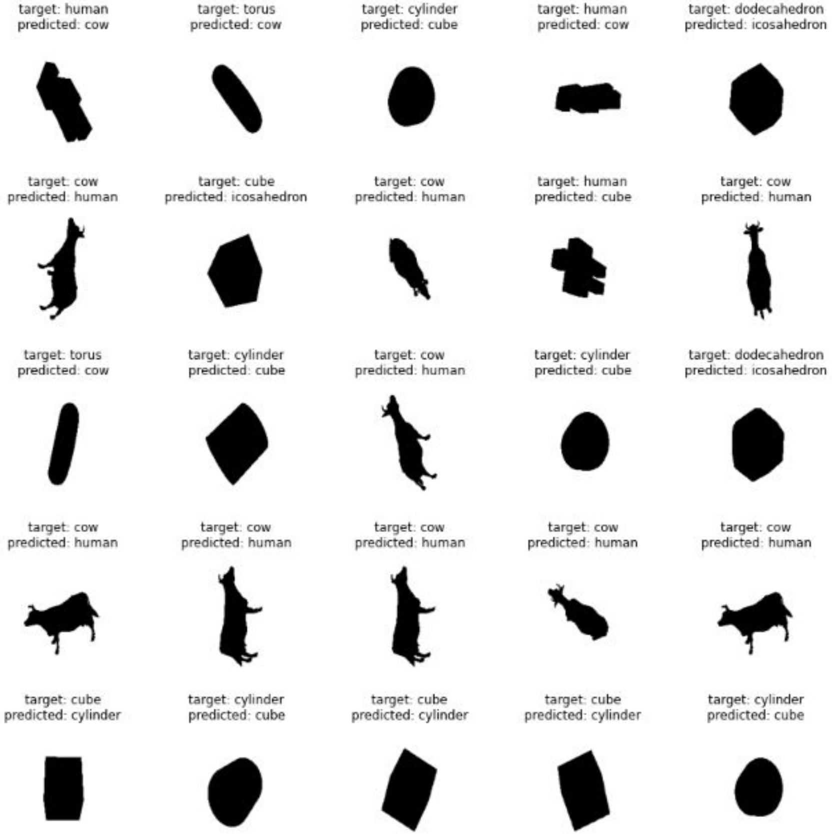


Full shapes

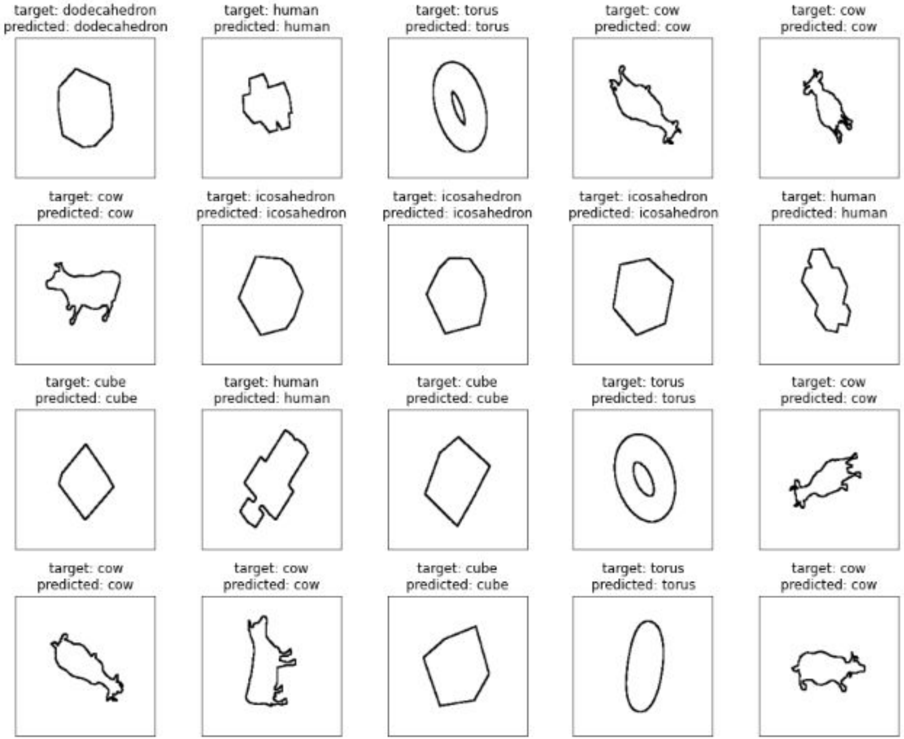


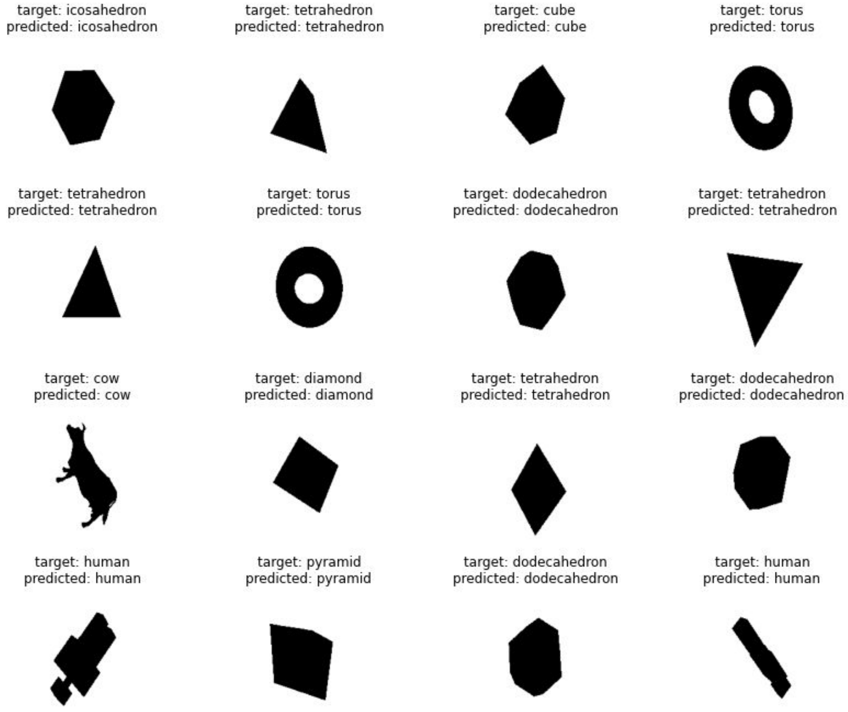
Wrong predictions





Correct predictions





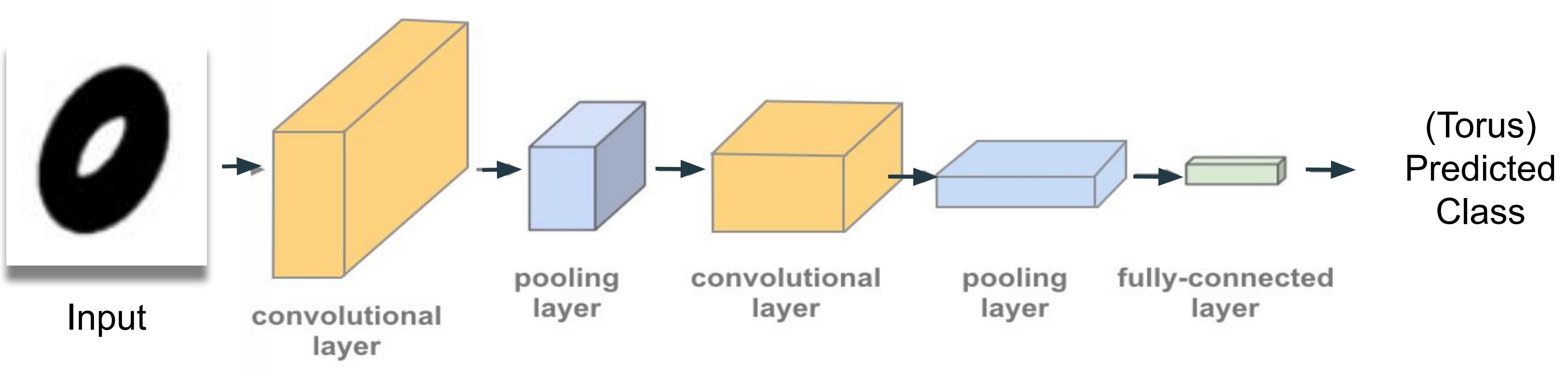
Conclusion for DNN

-full shapes perform better than edges

-good results for 7 shapes

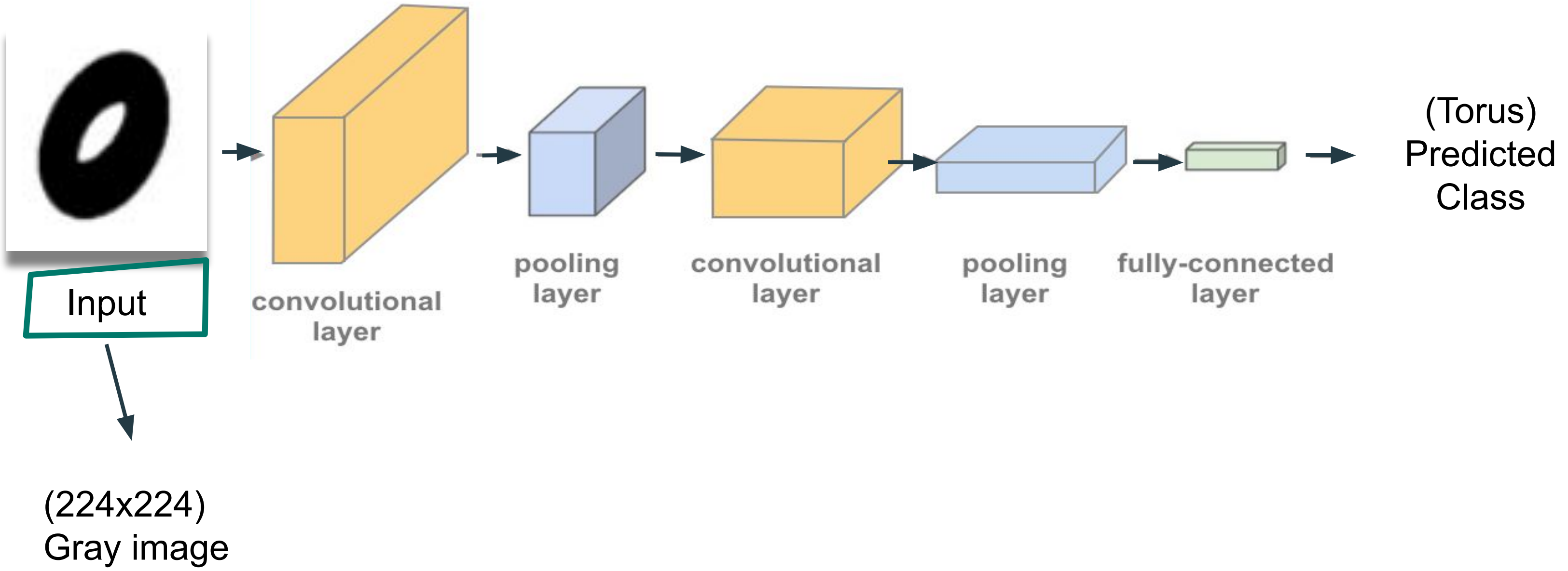
Convolutional Neural Network

CNN



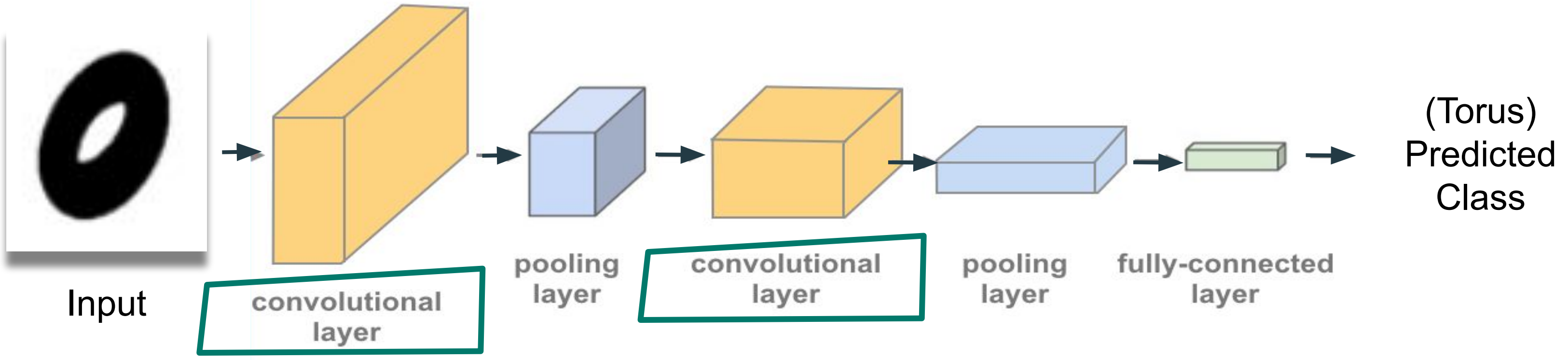
Input
convolutional layer
pooling layer
convolutional layer
pooling layer
fully-connected layer
(Torus) Predicted Class

CNN

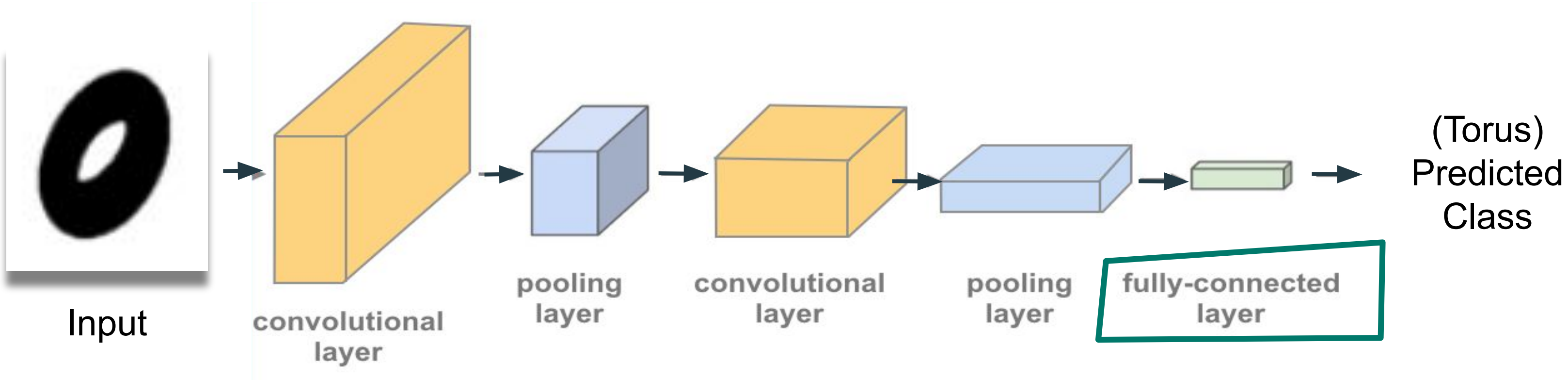


Input
convolutional layer
pooling layer
convolutional layer
pooling layer
fully-connected layer
(Torus) Predicted Class
(224x224) Gray image

CNN

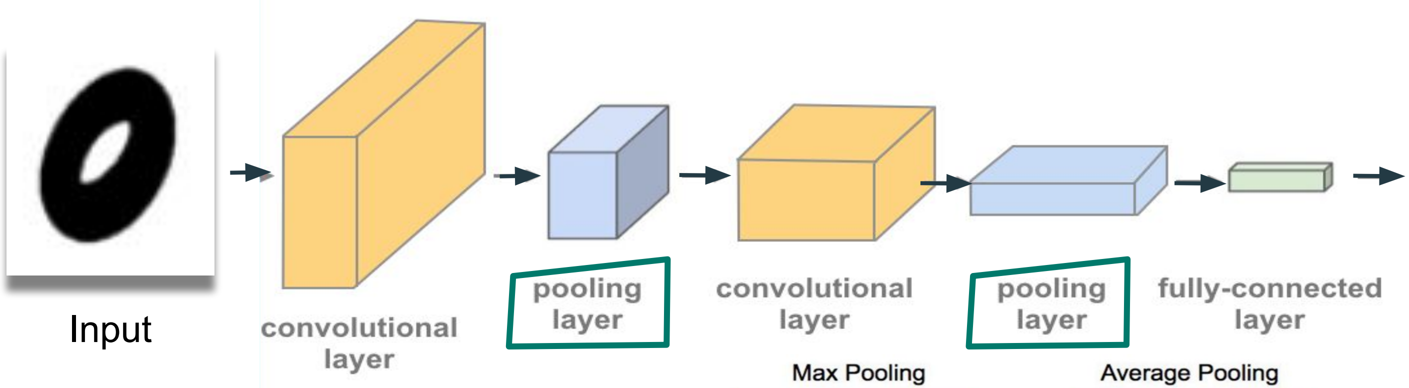

Input
convolutional layer
pooling layer
convolutional layer
pooling layer
fully-connected layer
(Torus) Predicted Class

CNN

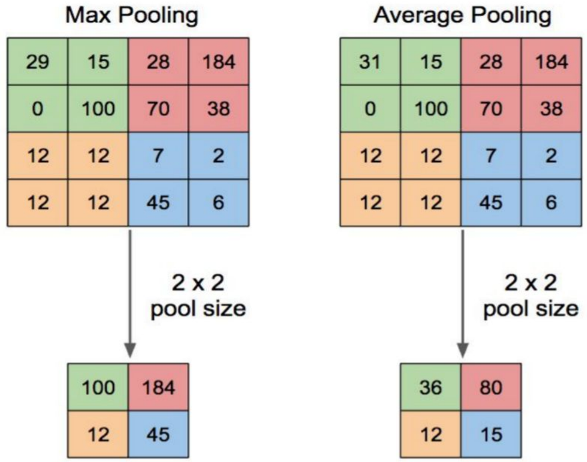


using Softmax layer (in the last layer of CNN, FC) for our multi-classification

CNN



(Torus)
Predicted
Class



We will use this later

reducing the computational complexity required to process the huge volume of data linked to an image.

Source:
https://towardsdatascience.com/convolution-neural-network-for-image-processing-using-keras-dc3429056306

CNN Results

We used various Architectures (VGG, LeNet,..) with different channels trends before any regularization.

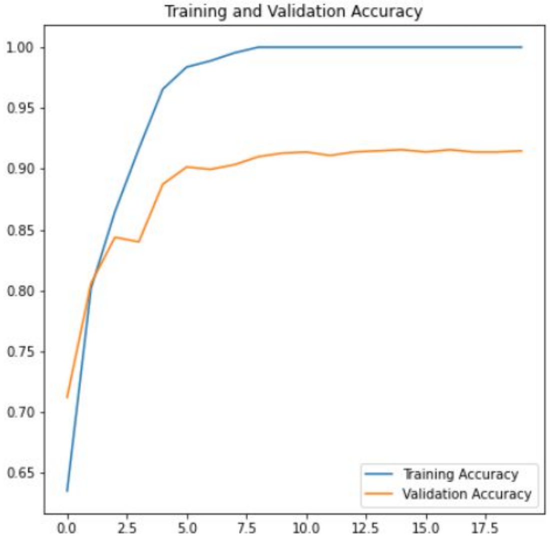
```
model.compile(optimizer = tf.optimizers.Adam() , loss=keras.losses.categorical_crossentropy, metrics = ['accuracy'])
```

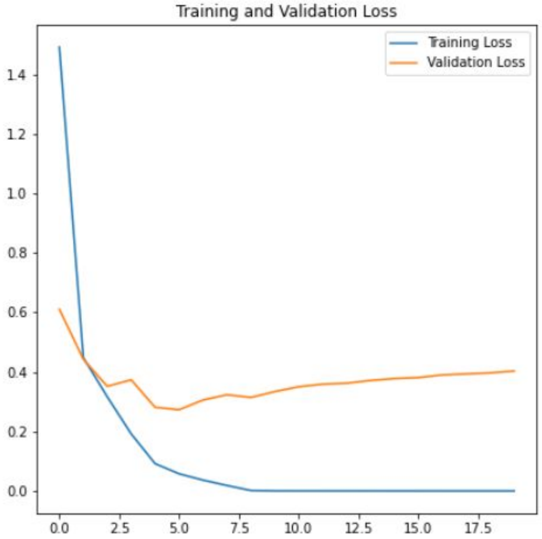
Learning_rate=0.01   batch_size = 32   epochs=20

CNN Results

VGG-like

```
Train loss: 3.516e-05
Train accuracy: 1.0
Test loss: 0.4031
Test accuracy: 0.9148
```



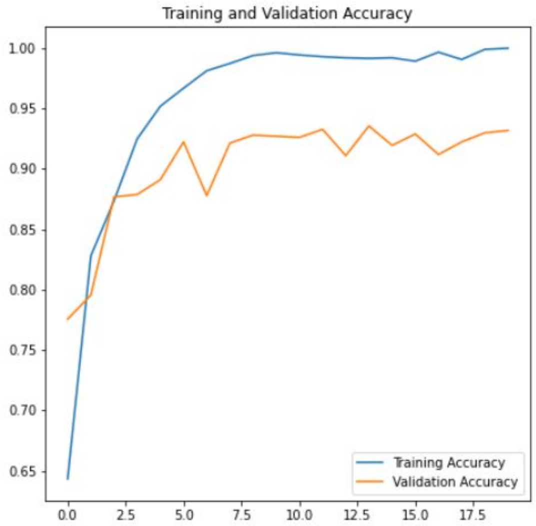



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_18 (Conv2D)           (None, 200, 200, 16)      160
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 200, 200, 16)      2320
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 100, 100, 16)      0
_________________________________________________________________
conv2d_20 (Conv2D)           (None, 98, 98, 32)        4640
_________________________________________________________________
conv2d_21 (Conv2D)           (None, 96, 96, 32)        9248
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 48, 48, 32)        0
_________________________________________________________________
flatten_4 (Flatten)          (None, 73728)             0
_________________________________________________________________
dense_8 (Dense)              (None, 512)               37749248
_________________________________________________________________
dense_9 (Dense)              (None, 4)                 2052
=================================================================
Total params: 37,767,668
Trainable params: 37,767,668
Non-trainable params: 0
_________________________________________________________________
```

CNN Results

```
Train loss: 0.0003416
Train accuracy: 1.0
Test loss: 0.2841
Test accuracy: 0.9318
```

LeNet

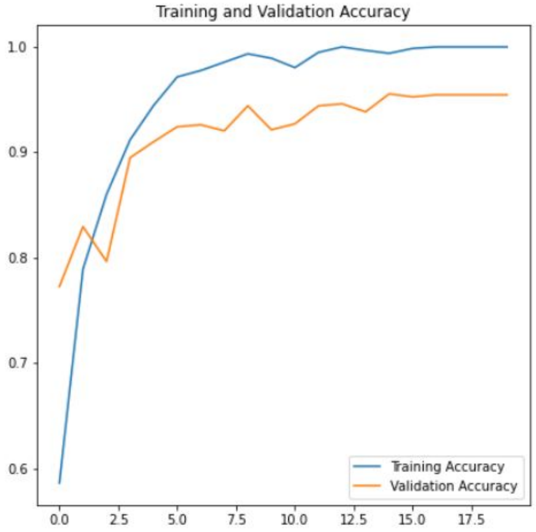


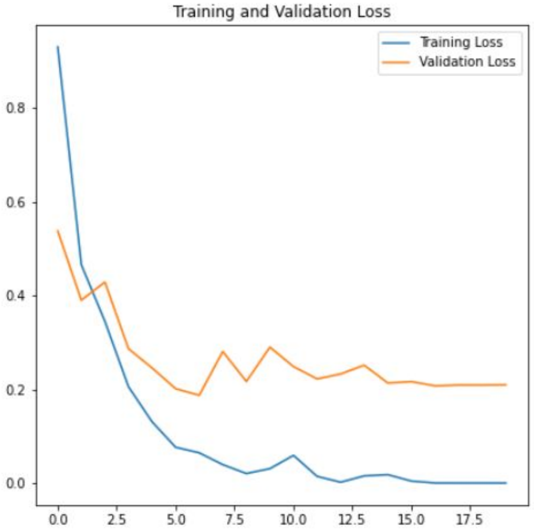

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_22 (Conv2D)           (None, 196, 196, 6)       156
_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 98, 98, 6)         0
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 94, 94, 16)        2416
_________________________________________________________________
max_pooling2d_16 (MaxPooling (None, 47, 47, 16)        0
_________________________________________________________________
flatten_5 (Flatten)          (None, 35344)             0
_________________________________________________________________
dense_10 (Dense)             (None, 120)               4241400
_________________________________________________________________
dense_11 (Dense)             (None, 84)                10164
_________________________________________________________________
dense_12 (Dense)             (None, 4)                 340
=================================================================
Total params: 4,254,476
Trainable params: 4,254,476
Non-trainable params: 0
_________________________________________________________________
```

CNN Results

Conv-Conv-Pool / 32-64-64 (highest Accuracy)

```
Train loss: 2.096e-05
Train accuracy: 1.0
Test loss: 0.2095
Test accuracy: 0.9545
```



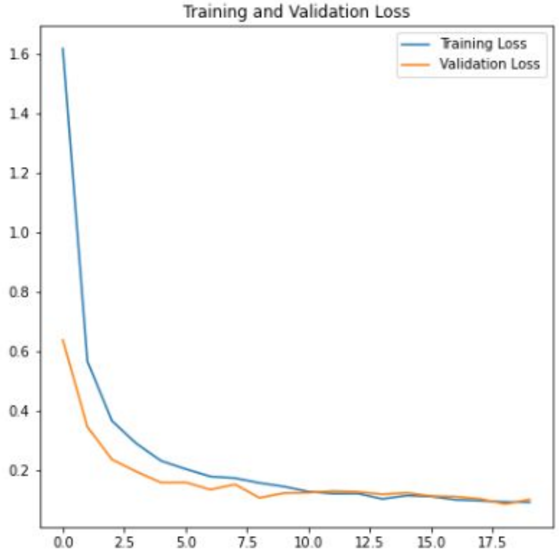



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 200, 200, 32)      320
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 198, 198, 32)      9248
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 99, 99, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 99, 99, 64)        18496
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 97, 97, 64)        36928
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 48, 48, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 48, 48, 64)        36928
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 46, 46, 64)        36928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 23, 23, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 33856)             0
_________________________________________________________________
dense (Dense)                (None, 512)               17334784
_________________________________________________________________
dense_1 (Dense)              (None, 10)                5130
=================================================================
Total params: 17,478,762
Trainable params: 17,478,762
Non-trainable params: 0
```

CNN Results

Conv-Conv-Pool-Conv-Conv-Pool / 32-64-64 (highest Accuracy)

```
Train loss: 0.03076
Train accuracy: 0.9987
Test loss: 0.1034
Test accuracy: 0.9758
```



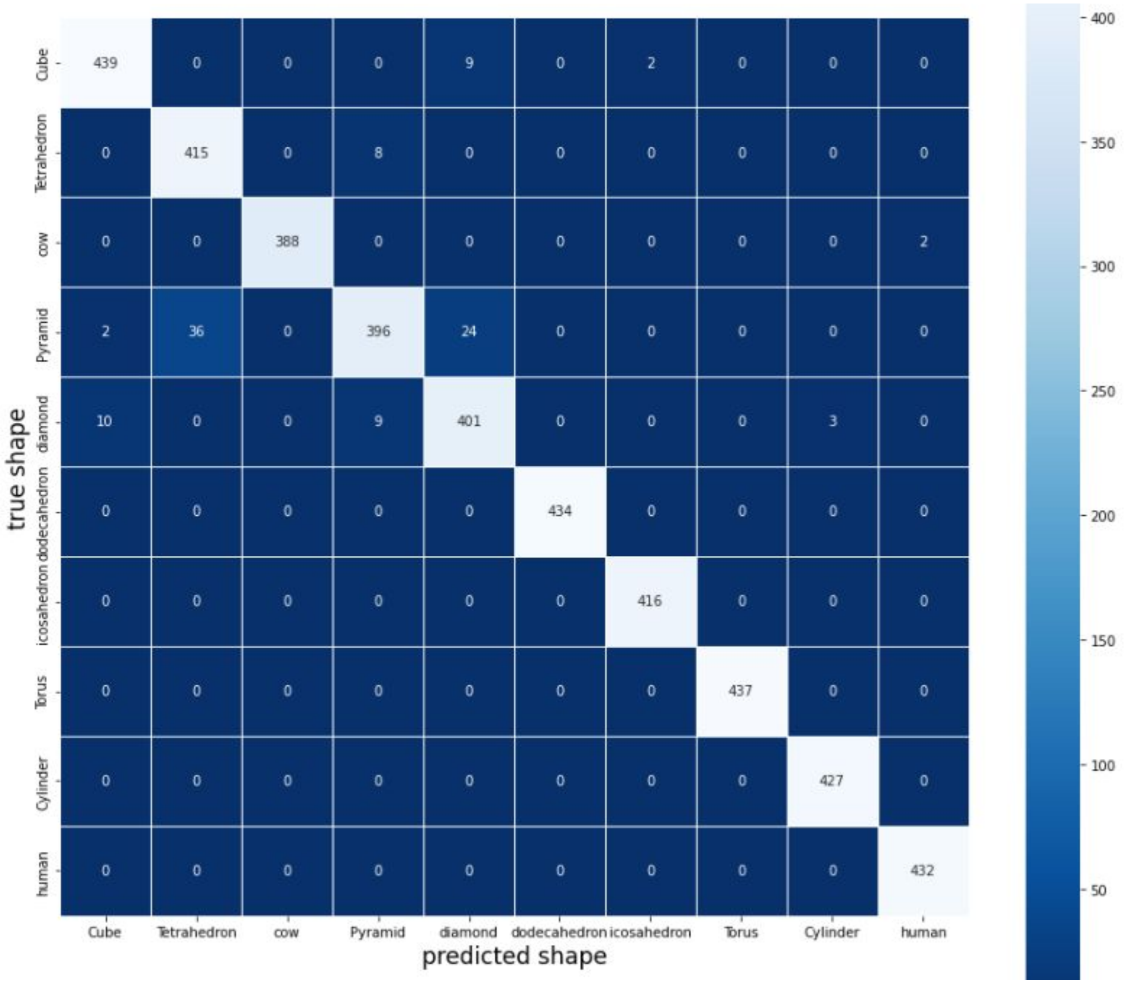
Now using regularization (dropout, L2)

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 200, 200, 32)      320
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 198, 198, 32)      9248
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 99, 99, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 99, 99, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 99, 99, 64)        18496
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 97, 97, 64)        36928
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 48, 48, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 48, 48, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 48, 48, 64)        36928
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 46, 46, 64)        36928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 23, 23, 64)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 23, 23, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 33856)             0
_________________________________________________________________
dense (Dense)                (None, 512)               17334784
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                5130
=================================================================
Total params: 17,478,762
Trainable params: 17,478,762
Non-trainable params: 0
_________________________________________________________________
```

CNN Results

Test accuracy: 0.9758

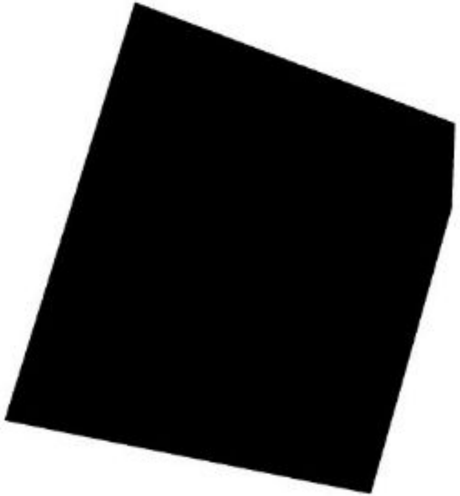
	precision	recall	f1-score	support
Pyramid	0.94	0.86	0.90	433
Tetrahedron	0.94	0.95	0.95	406
cow	1.00	0.99	1.00	450
Cube	0.98	0.99	0.98	415
Torus	1.00	1.00	1.00	436
diamond	0.92	0.98	0.95	412
dodecahedron	1.00	1.00	1.00	400
human	0.99	1.00	1.00	446
Cylinder	1.00	1.00	1.00	469
icosahedron	1.00	1.00	1.00	423
accuracy			0.98	4290
macro avg	0.98	0.98	0.98	4290
weighted avg	0.98	0.98	0.98	4290



Challenge

Click to Present

what is the shape?



tinyurl.com/MLHO22

Diamond

CNN Results

Conv-Conv-Pool-Conv-Conv-Pool / 32-64-64 (highest Accuracy)

Some good predictions

target: Cube
predicted: Cube

target: Tetrahedron
predicted: Tetrahedron

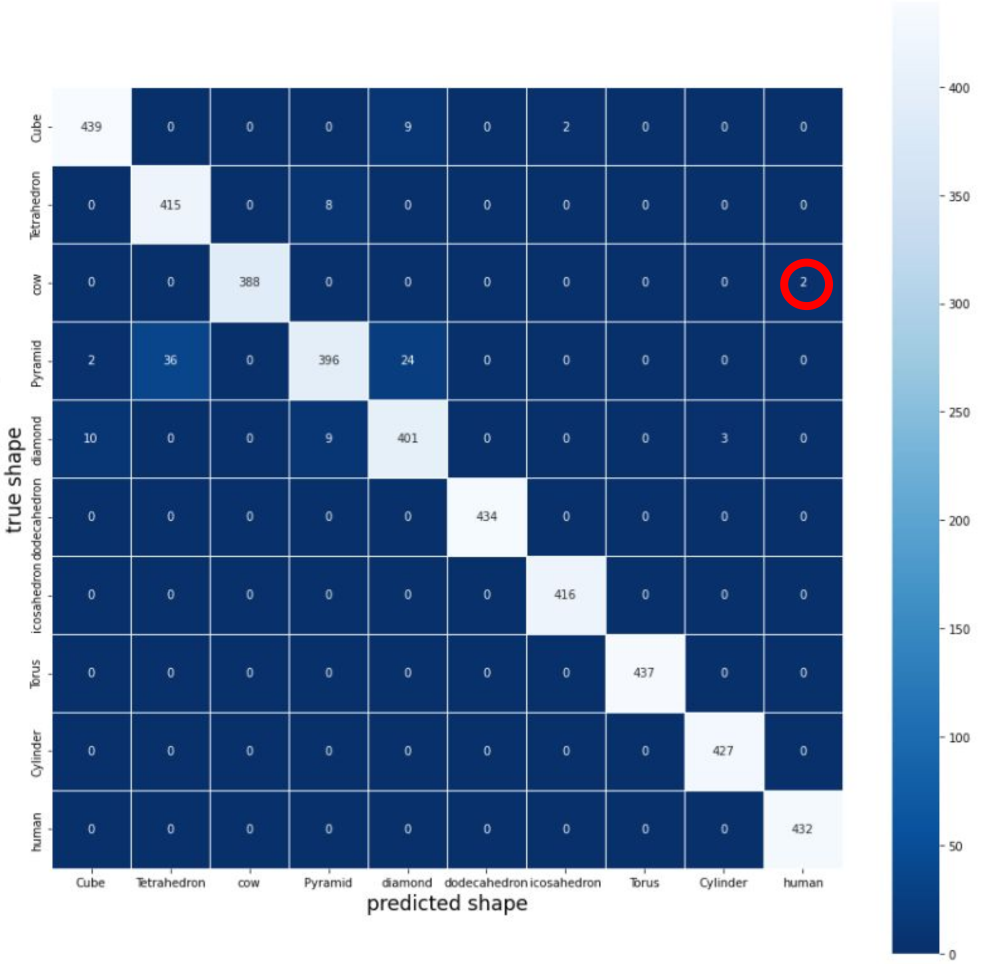
target: diamond
predicted: diamond

target: Tetrahedron
predicted: Tetrahedron

Some errors

target: cow
predicted: human

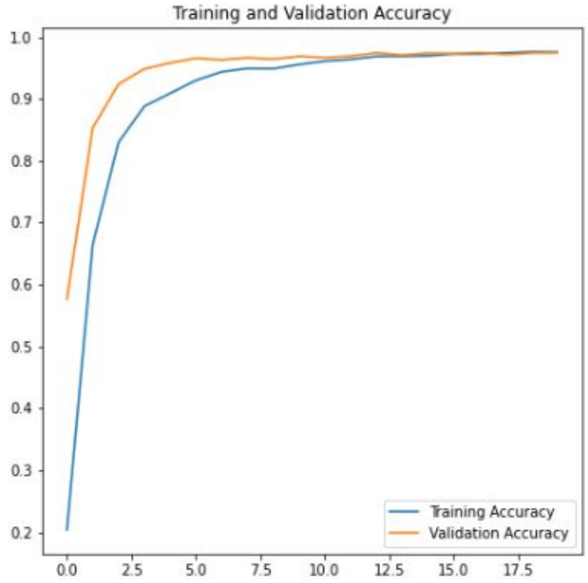
target: Pyramid
predicted: diamond

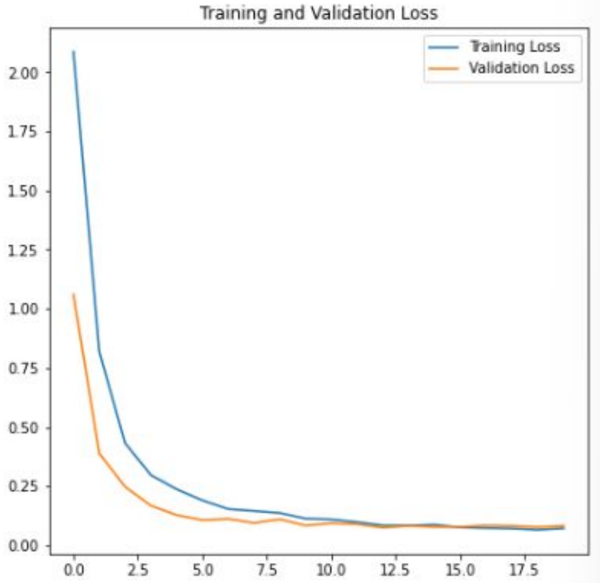


CNN Results (Edges)

Conv-Conv-Pool / 16-32-64

```
Train loss: 0.004168
Train accuracy: 0.9994
Test loss: 0.0821
Test accuracy: 0.9744
```



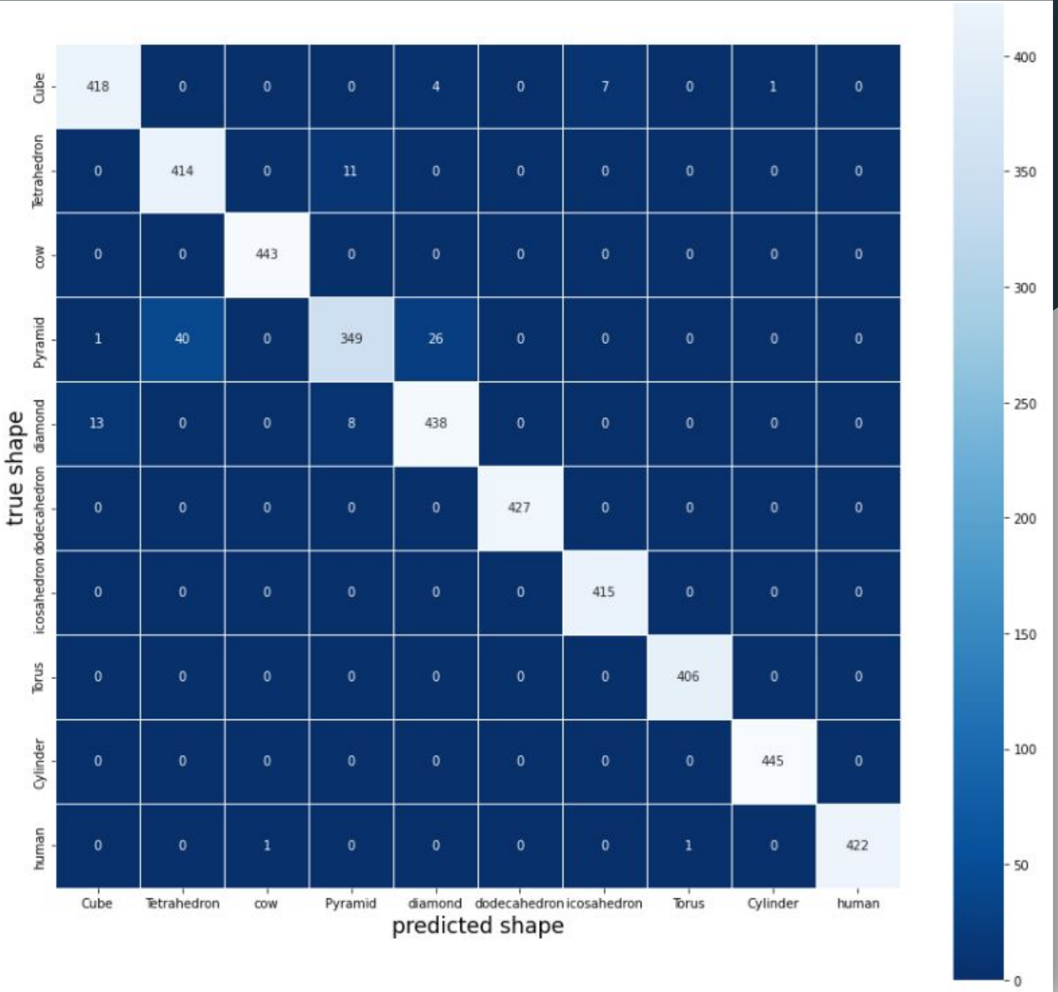



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)            (None, 200, 200, 16)      160
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 198, 198, 16)      2320
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 99, 99, 16)        0
_________________________________________________________________
dropout_4 (Dropout)          (None, 99, 99, 16)        0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 99, 99, 32)        4640
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 97, 97, 32)        9248
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 48, 48, 32)        0
_________________________________________________________________
dropout_5 (Dropout)          (None, 48, 48, 32)        0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 48, 48, 64)        18496
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 46, 46, 64)        36928
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 23, 23, 64)        0
_________________________________________________________________
dropout_6 (Dropout)          (None, 23, 23, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 33856)             0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               17334784
_________________________________________________________________
dropout_7 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                5130
=================================================================
Total params: 17,411,706
Trainable params: 17,411,706
Non-trainable params: 0
_________________________________________________________________
```

CNN Results (Edges)

	precision	recall	f1-score	support
Cube	0.97	0.97	0.97	420
Tetrahedron	0.97	0.95	0.96	444
cow	0.97	1.00	0.99	440
Pyramid	0.92	0.92	0.92	406
diamond	0.94	0.95	0.94	425
dodecahedron	1.00	1.00	1.00	411
icosahedron	0.99	1.00	1.00	461
Torus	0.99	1.00	0.99	426
Cylinder	1.00	1.00	1.00	417
human	1.00	0.96	0.98	440
accuracy			0.97	4290
macro avg	0.97	0.97	0.97	4290
weighted avg	0.97	0.97	0.97	4290

```
Train loss: 0.004168
Train accuracy: 0.9994
Test loss: 0.0821
Test accuracy: 0.9744
```



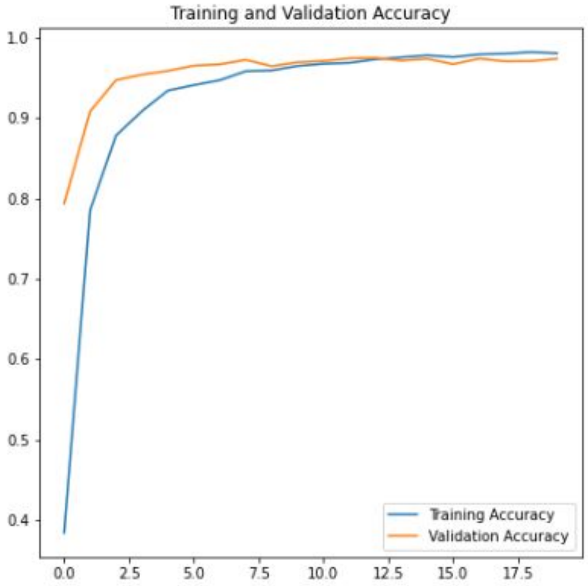
CNN Results (Solid shadows again)

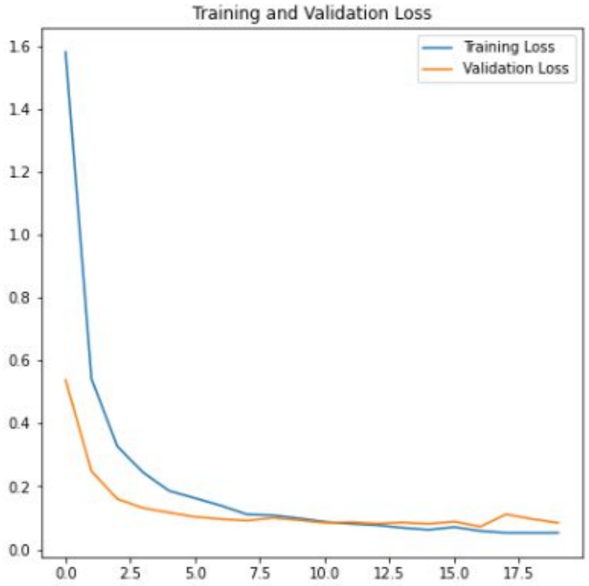
Conv-Conv-Pool-Conv-Conv-Pool / 16-32-64 (drop out only) ALL shapes(msh far2a kteer)

```
Train loss: 0.00195
Train accuracy: 0.9998
Test loss: 0.09294
Test accuracy: 0.9755
```

Layer (type)	Output Shape	Param #
conv2d_6 (Conv2D)	(None, 200, 200, 16)	160
conv2d_7 (Conv2D)	(None, 198, 198, 16)	2320
max_pooling2d_3 (MaxPooling2	(None, 99, 99, 16)	0
dropout_4 (Dropout)	(None, 99, 99, 16)	0
conv2d_8 (Conv2D)	(None, 99, 99, 32)	4640
conv2d_9 (Conv2D)	(None, 97, 97, 32)	9248
max_pooling2d_4 (MaxPooling2	(None, 48, 48, 32)	0
dropout_5 (Dropout)	(None, 48, 48, 32)	0
conv2d_10 (Conv2D)	(None, 48, 48, 64)	18496
conv2d_11 (Conv2D)	(None, 46, 46, 64)	36928
max_pooling2d_5 (MaxPooling2	(None, 23, 23, 64)	0
dropout_6 (Dropout)	(None, 23, 23, 64)	0
flatten_1 (Flatten)	(None, 33856)	0
dense_2 (Dense)	(None, 512)	17334784
dropout_7 (Dropout)	(None, 512)	0
dense_3 (Dense)	(None, 10)	5130

Total params: 17,411,706
Trainable params: 17,411,706
Non-trainable params: 0



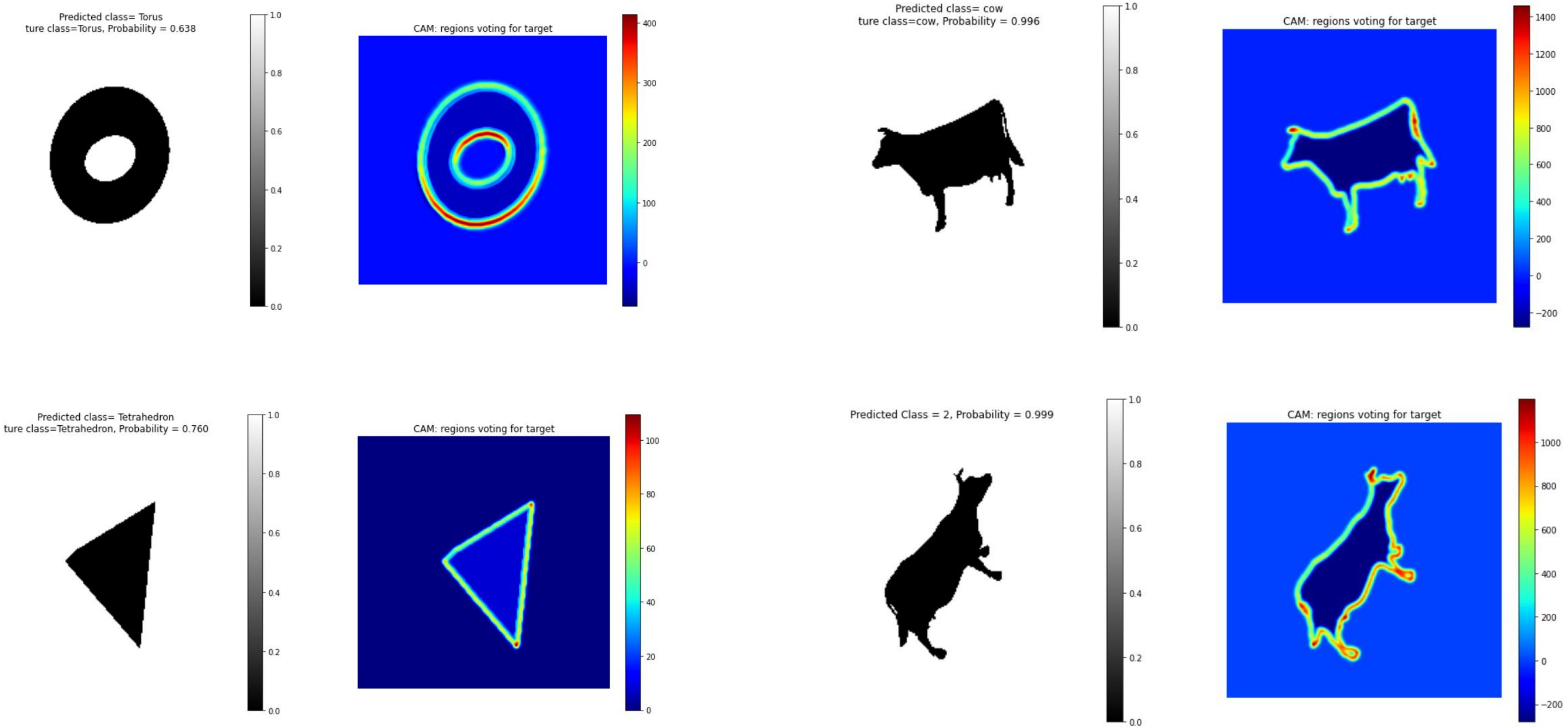


CAM

```
model = Sequential()
model.add(Conv2D(16,input_shape=input_shape,kernel_size=(3,3),activation='relu',padding='same'))
model.add(Conv2D(32,kernel_size=(3,3),activation='relu',padding='same'))
model.add(Conv2D(64,kernel_size=(3,3),activation='relu',padding='same'))
model.add(Conv2D(128,kernel_size=(3,3),activation='relu',padding='same'))
model.add(GlobalAveragePooling2D())
model.add(Dense(len(labels),activation='softmax'))
```

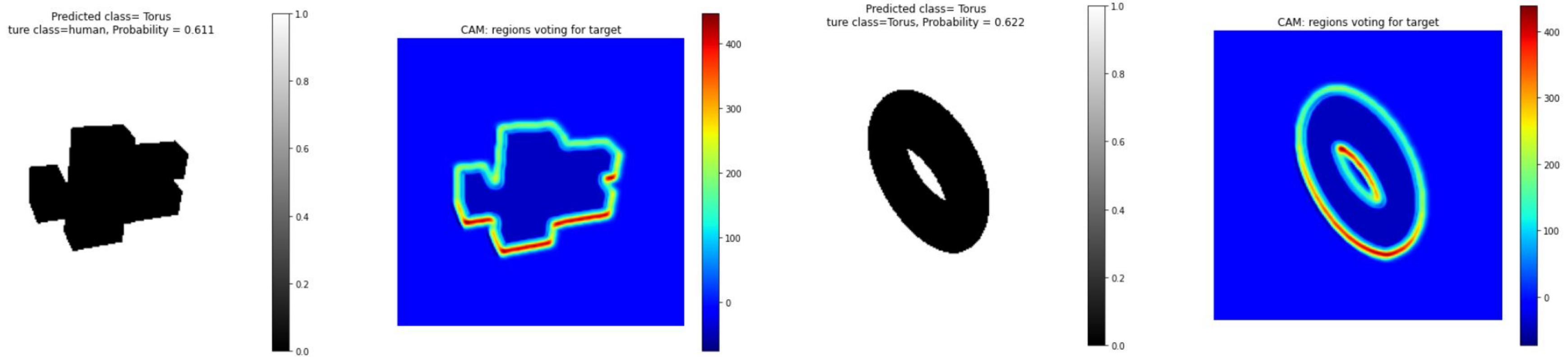

```
Train loss: 0.7704
Train accuracy: 0.6635
Test loss: 0.764
Test accuracy: 0.6613
```

CAM



Predicted class= Torus
ture class=Torus, Probability = 0.638
1.0
0.8
0.6
0.4
0.2
0.0
CAM: regions voting for target
400
300
200
100
0
Predicted class= cow
ture class=cow, Probability = 0.996
1.0
0.8
0.6
0.4
0.2
0.0
CAM: regions voting for target
1400
1200
1000
800
600
400
200
0
−200
Predicted class= Tetrahedron
ture class=Tetrahedron, Probability = 0.760
1.0
0.8
0.6
0.4
0.2
0.0
CAM: regions voting for target
100
80
60
40
20
0
Predicted Class = 2, Probability = 0.999
1.0
0.8
0.6
0.4
0.2
0.0
CAM: regions voting for target
1000
800
600
400
200
0
−200

CAM


Predicted class= Torus
ture class=human, Probability = 0.611
CAM: regions voting for target
Predicted class= Torus
ture class=Torus, Probability = 0.622
CAM: regions voting for target

Conclusion and Outlook

- CNN proved itself capable of better image classifications than DNN
- The problem is underdetermined because some shadows of different shapes are the same for different orientations
- Complicated shapes are easier to distinguish
- The project can be extended to cover more shapes and other lighting conditions

Thank you for your attention

Questions?