In GOD WE TRUST

Deep Learning Course

2021-2022 Fall Semester

HW02

CNN

Omid Sharafi
(400201518)

Instructor:
Dr.Emad Fatemizadeh

December 29, 2021

Sharif
University
of
Technology

Department of Electrical Engineering

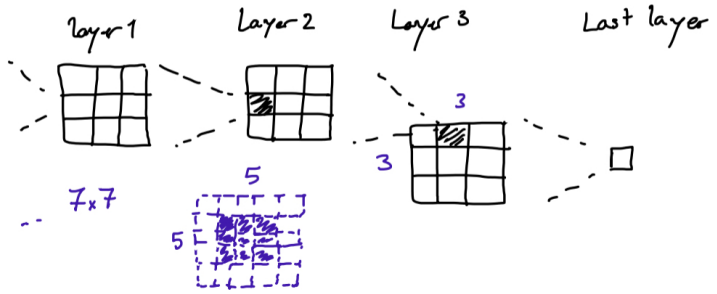
Contents

1 Theoretical Problems

1.1 Network Design

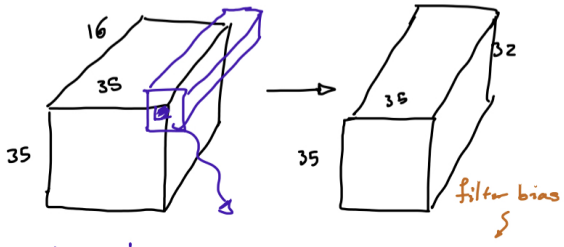
Problem ①: Network Design

A) $9 \times 9 = 81$



B) By adding pooling layer, we are somehow summarizing the previous C layer So that it helps us to reduce the network parameters & calculation load. On the other hand we are selecting the most important features & pooling will increase the network robustness against input movement.

C)



parameter number = $32 \times 16 \times 3 \times 3 + 32$

D) $[227 \times 227 \times 3] \xrightarrow{Conv1 = (256, 5, 5)} [227 \times 227 \times 256] \xrightarrow{Conv2 = (128, 5, 5)} [227 \times 227 \times 128]$

$P = 256 \times 3 \times 5 \times 5 + 256$ $\quad P = 128 \times 5 \times 5 \times 256 + 128$

Max Pooling (5, 5, stride 2x2) $[113 \times 113 \times 128] \xrightarrow{FC} 512 \xrightarrow{FC} 100$

$P = 0$ $\quad P = 113 \times 113 \times 128 \times 512 + 512$ $\quad P = 512 \times 100 + 10$

Sum_of_parameters = 19,456 + 819,328 + 836,829,696 + 51,210 ≈ 840M !

The main problem of this network is its large parameter number. As you can see, the network layer sizes are: 227×227 → 227×227 → 113×113 → flatten !
This design is not good ☹ we should design a network like a pyramid that has reduction in size step by step by having strides in C layers & max pooling.

My design:

[227×227×3] —Conv1: (256×5×5)→ [227×227×256] —Conv2: (128×5×5, stride 2×2)→ [113×113×128] —max pool (5×5, stride 2×2)→ [56×56×128] —Conv3: (128×5×5, stride 2×2)→ [27×27×128] —Max Pooling (5×5, stride 2×2)→ [13, 13, 128] —FC→ 256 —FC→ 256 —FC→ 100

$P = (256\times5\times5\times3+256) + (128\times5\times5\times256+128) + (128\times5\times5\times128+128) + (128\times5\times5\times128+128) + (128\times5\times5\times128+128) + (256\times128\times13\times13+256) + (256\times256+256) + (100\times256+100) \simeq 7.5M$

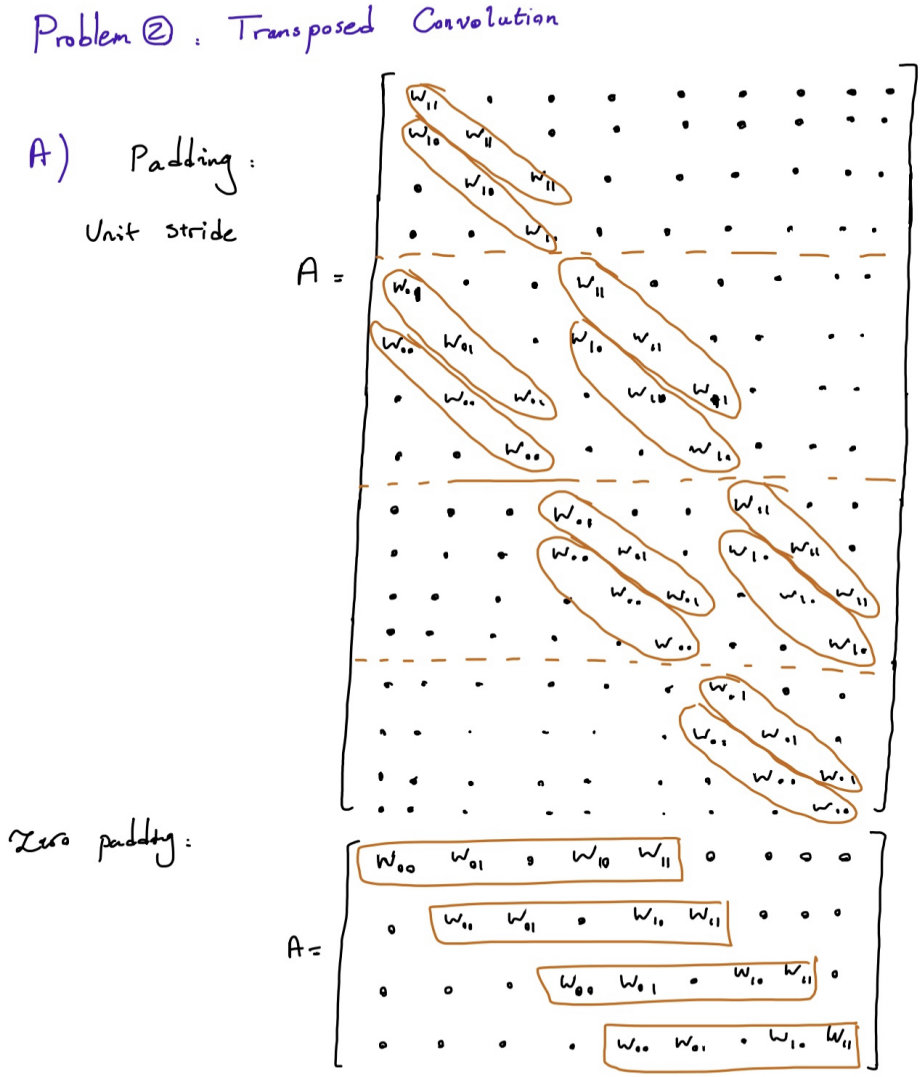
(≈ 20K, ≈ 800K, ≈ 400K, ≈ 400K, ≈ 400K, 5.5M, 70K, 25K)

Nice!

E) The general function approximation just say that we can do this by two layers, but it does not talk about how many neurons we need at each layer for doing so. As we have seen in the course resources, increasing the layers of the network, make it more efficient than just increasing number of neurons in a sigle layer.

F) In EPB algorithm, for gradient calculation we are multiplying lots of gradients. As the network becomes deeper, if these gradints are less than 1 ther result converges to zero that is gradient vanishing. If these gradients are more than 1 then the result at deep layers converges to ∞ We call this state gradient expladin.

1.2 Transposed Convolution



B) Masked Convolution : This kind of convolution is just masks some especiall pixels of the input. So it can help us to do something like dropout & on the other hand we are reducing the network parameters. So it can help us to improve the network traing speed and preventing overfitting.

C) Separable convolution : In spatial seprable convolution we separate the kernel to small kernels so that we are making our model more powerful and also decreasing the paramater numbers. for example :

5×5 (25 param) $\rightarrow$ $3 \times 3 + 3 \times 3$ (18 params)

3×3 (9 params) $\rightarrow$ $1 \times 3 + 3 \times 1$ (6 params)

Also in depthwise separable convolution we separate the 3D kernel in different dimentions.

1.3 Object Detection (+)

Problem ③ Object Detection :

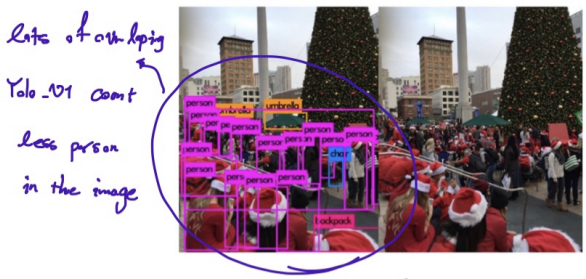
A) { Yolo_v1 : B = 2 , C = 300 ⟹ Bounding Box (x, y, w, h, confidence) → 2 × 5 + 300 = 310

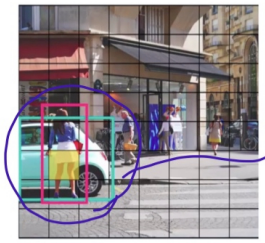
Yolo_v3 : For this network we have 3 (multi resolution) outputs

But in each resolution at the last layer we have 3 Anchor Boxes.

So : The last layer depth = 3 × (5 + 300) = 915

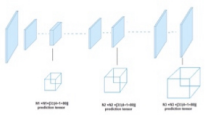
B) The problem of Yolo_v1 is that it can not detect small objects or objects with overlap. This problem comes from that for each cell in Yolo_v1 we assign a single object to it and we work with centers of the objects.





But in Yolo_v2 instead of just one confidence & probability calculation we have term of Anchor Boxes for each cell of the grid and it solve the problem of object detection with over laping.

C) In Yolo_v3 as said in part A, we have multi resolution analysis in sizes 13×13 , 26×26 , 52×52 . We have also 3 anchor boxes so that we have 3×3 = 9 totally that can be another reson of improvment.



2 MLP,CNN

2.1 Batch size

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 3072)              0

 gaussian_noise (GaussianNoi  (None, 3072)             0
 se)

 dense (Dense)               (None, 1024)              3146752

 gaussian_dropout (GaussianD  (None, 1024)             0
 ropout)

 dense_1 (Dense)             (None, 256)               262400

 dropout (Dropout)           (None, 256)               0

 dense_2 (Dense)             (None, 10)                2570

=================================================================
Total params: 3,411,722
Trainable params: 3,411,722
Non-trainable params: 0
_________________________________________________________________
```

Figure 1: Network Summary

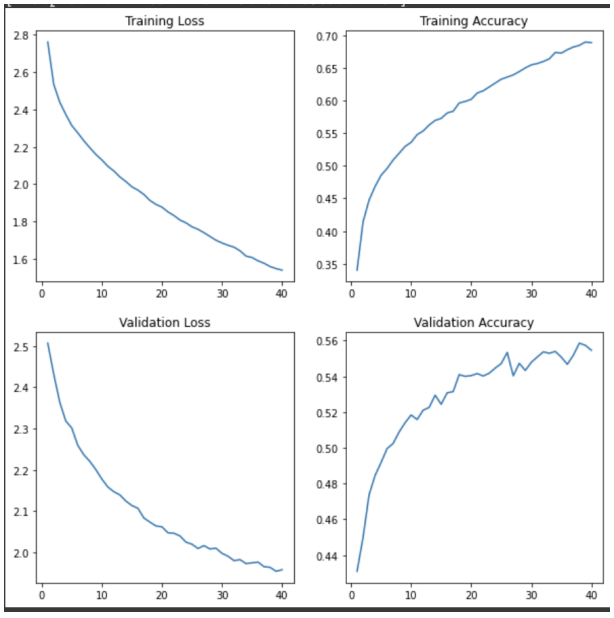


Figure 2: Batch size = 32, Run time per each epoch = 9s

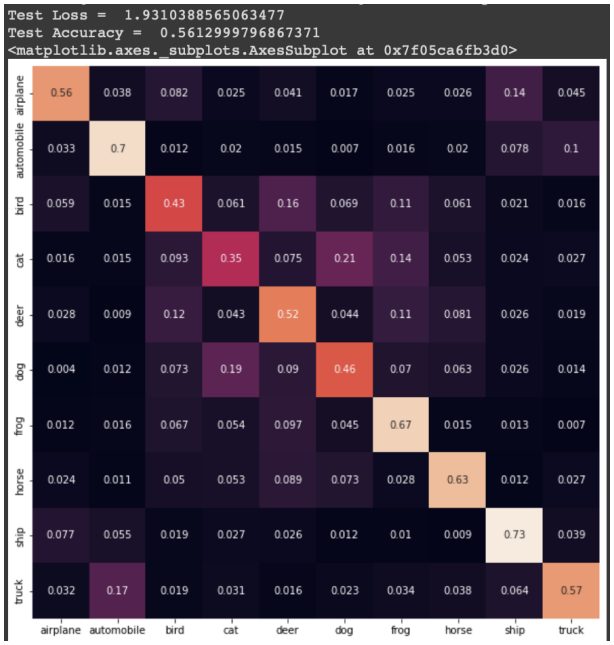


Figure 3: Batch size = 32

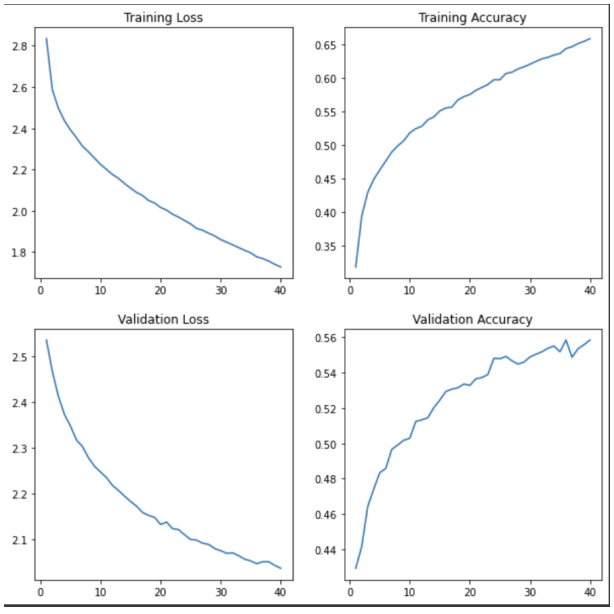


Figure 4: Batch size = 64, Run time per each epoch = 5s

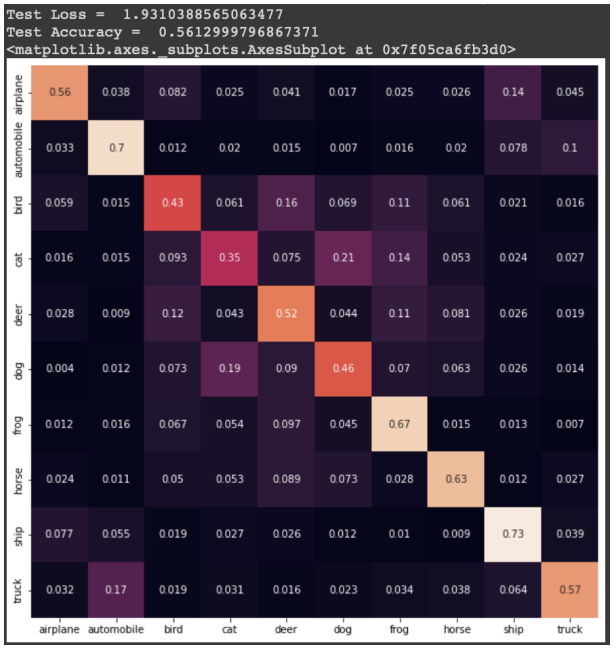


Figure 5: Batch size = 64

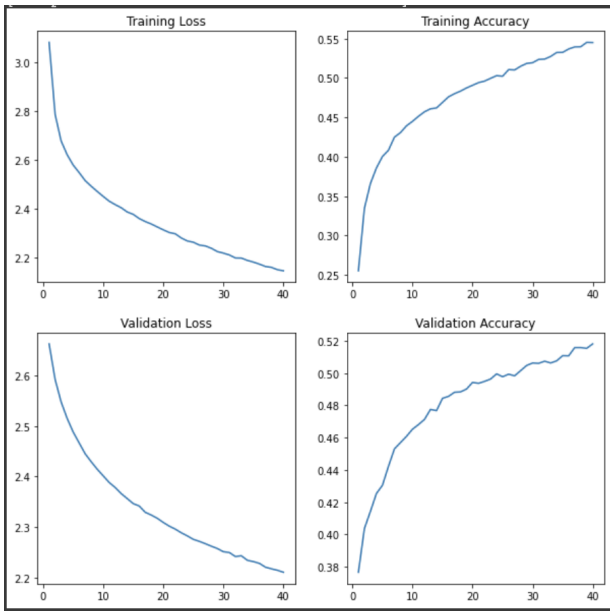


Figure 6: Batch size = 256, Run time per each epoch = 2s

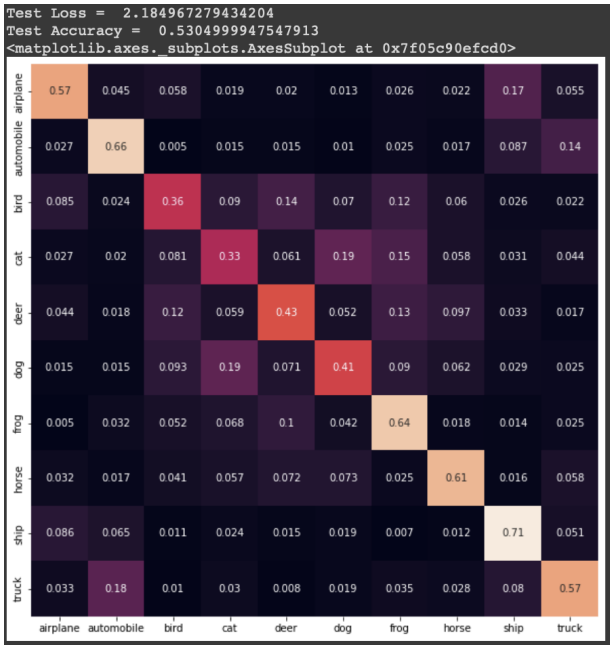


Figure 7: Batch size = 256

As you can see in the results, increasing batch size reduce the accuracy and run time.

2.2 Activation Function

TanH function is gradient smooth, bounded, and zero at origin, but it causes vanishing gradient. ReLU activation function is non-linear, zero at origin, and computationally efficient, but it has no gradient at origin and its completely flat and zero for negative numbers. The results show that ReLU works better than TanH in this problem.

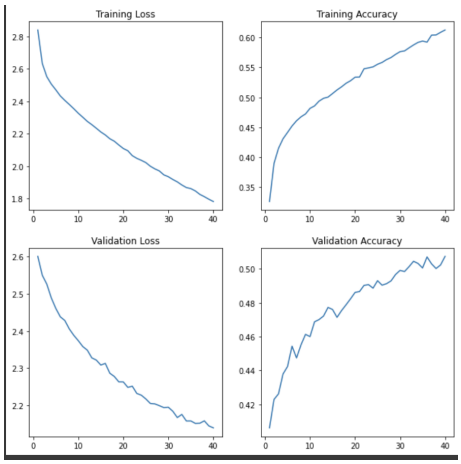


Figure 8: Activation Function TanH

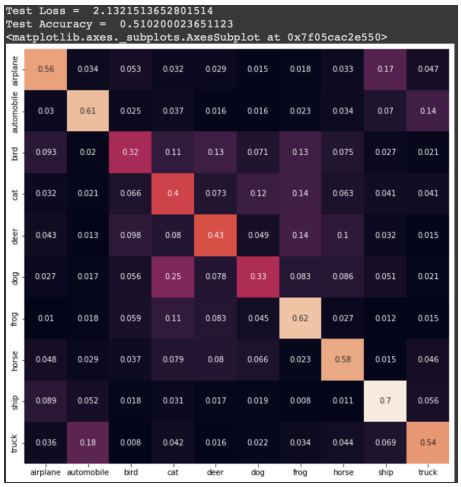


Figure 9: Activation Function TanH

2.3 Error Function

For this multi classification problem we should use cross entropy. As you can see below, MSE not work good for our case and it has more error.

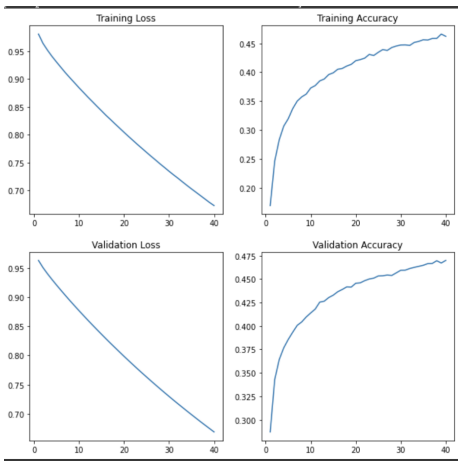


Figure 10: Loss Function MSE

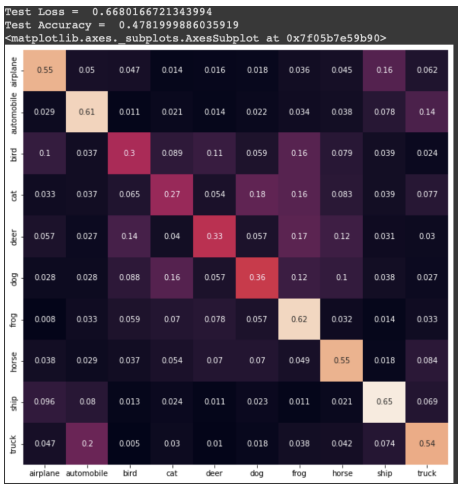


Figure 11: Loss Function MSE

2.4 Adam optimizer

The results show that SGD works better that Adam. Although SGD may go to its local minimum but it is more generalized and works better here.

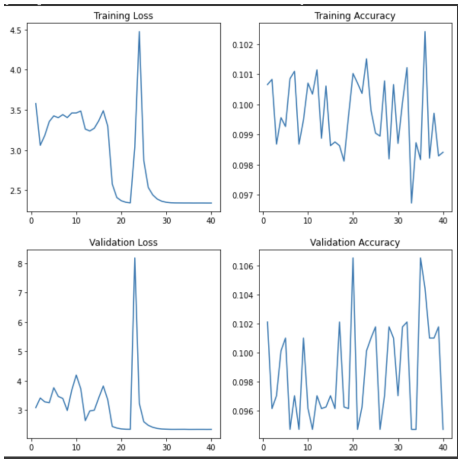


Figure 12: Adam optimizer

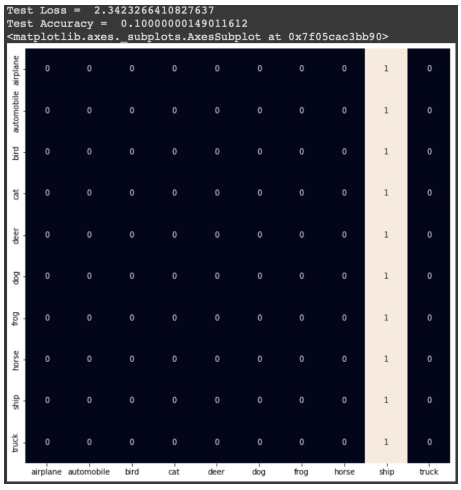


Figure 13: Adam optimizer

As you can see, Adam do not converge in our problem.

2.5 Best parameters

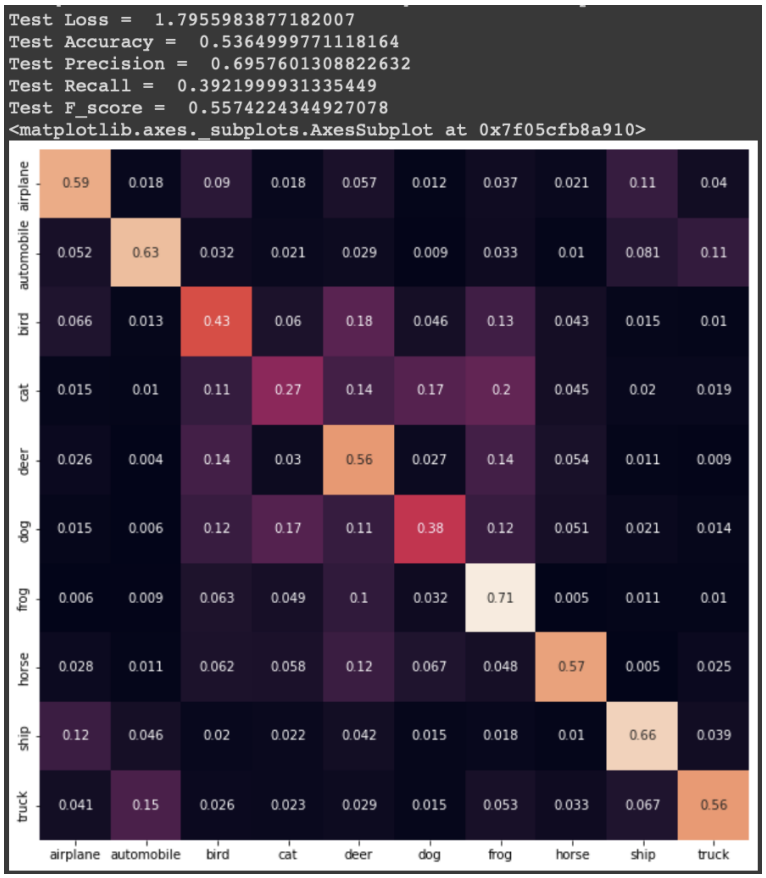


Figure 14: F-Score, Recall, and Precision

2.6 Unbalanced data set

When the data set is unbalanced, we have prediction problem in that categories. We saw this problem in previous assignment in confusion matrix.

For solving this problem we should use data augmentation for categories that suffer from lack of data to make a balanced data set for all categories images.

2.7 Test unbalanced data set

In this part we are removing half of airplanes and birds data. For solving this problem we know that we can flip images horizontally and still that image category remains.

So for solving this problem I have removed half of the images in the airplane and bird data set and added flip form of them to the data set. The network result for new data set is shown below :

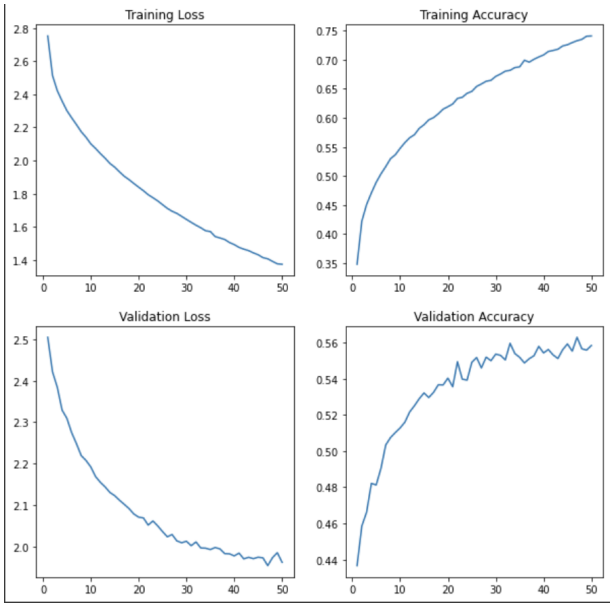


Figure 15: Network result after augmentation of unbalanced data set

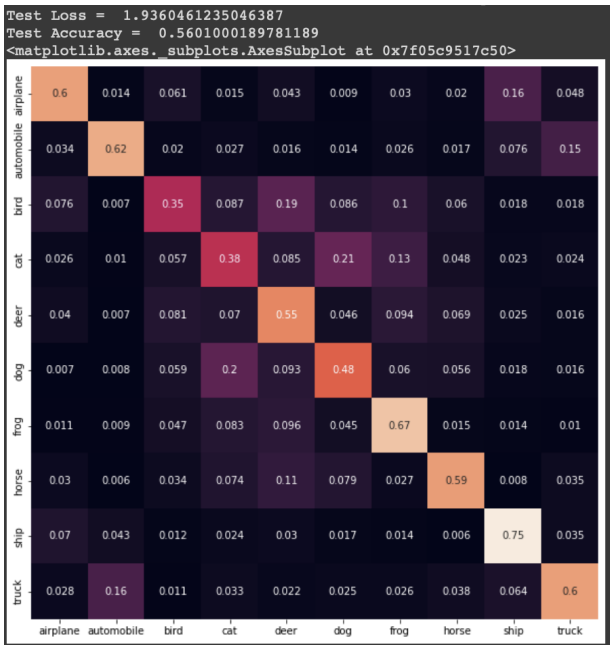


Figure 16: Network result after augmentation of unbalanced data set

The result is good and near original data set after flipping unbalanced categories to make the data set balanced.

2.8 Add CNN

```
Model: "sequential_14"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_6 (Conv2D)           (None, 28, 28, 128)       9728

 conv2d_7 (Conv2D)           (None, 24, 24, 64)        204864

 conv2d_8 (Conv2D)           (None, 20, 20, 64)        102464

 flatten_11 (Flatten)        (None, 25600)             0

 gaussian_noise_11 (Gaussian  (None, 25600)            0
 Noise)

 dense_33 (Dense)            (None, 1024)              26215424

 gaussian_dropout_11 (Gaussi  (None, 1024)             0
 anDropout)

 dense_34 (Dense)            (None, 256)               262400

 dropout_11 (Dropout)        (None, 256)               0

 dense_35 (Dense)            (None, 10)                2570

=================================================================
Total params: 26,797,450
Trainable params: 26,797,450
Non-trainable params: 0
_________________________________________________________________
```

Figure 17: CNN network design

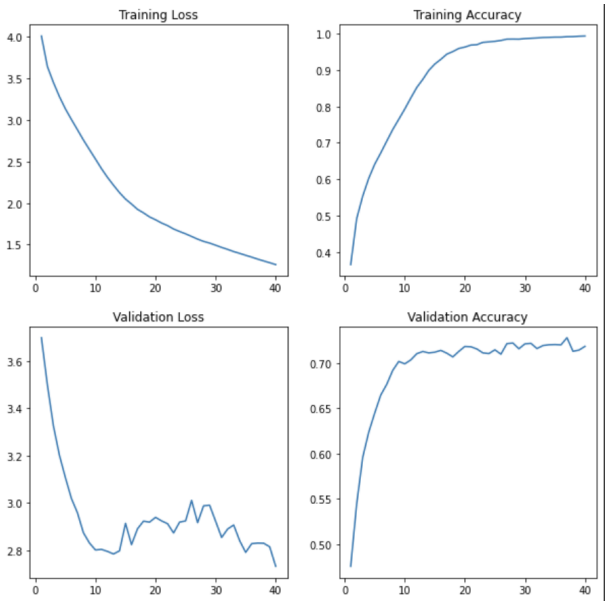


Figure 18: CNN result

We can see improvement in network performance.

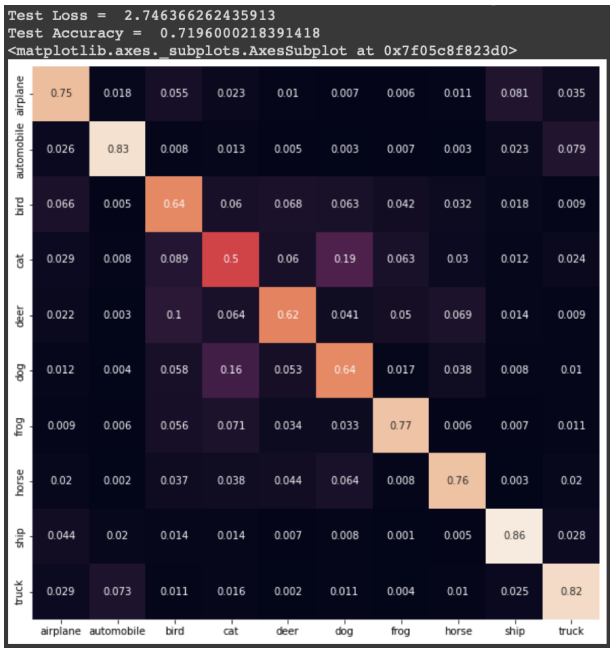


Figure 19: CNN result

2.9 Add Pooling and Batch normalization

Batch normalization helps each layer parameters to again normalize over a normal distribution so it helps to bound and normalize each layer output and standrize the learning process during EBP.

Pooling layers help us to reduce parameters of the network. We are selecting the most important features and reducing computational cost of calculation. We also improve network robustness against input movement.

```
Model: "sequential_18"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_12 (Conv2D)          (None, 28, 28, 128)       9728

 batch_normalization_3 (Batc  (None, 28, 28, 128)      512
 hNormalization)

 conv2d_13 (Conv2D)          (None, 24, 24, 64)        204864

 batch_normalization_4 (Batc  (None, 24, 24, 64)       256
 hNormalization)

 max_pooling2d_2 (MaxPooling  (None, 12, 12, 64)       0
 2D)

 conv2d_14 (Conv2D)          (None, 8, 8, 64)          102464

 batch_normalization_5 (Batc  (None, 8, 8, 64)         256
 hNormalization)

 max_pooling2d_3 (MaxPooling  (None, 4, 4, 64)         0
 2D)

 flatten_13 (Flatten)        (None, 1024)              0

 gaussian_noise_13 (Gaussian  (None, 1024)             0
 Noise)

 dense_39 (Dense)            (None, 1024)              1049600

 gaussian_dropout_14 (Gaussi  (None, 1024)             0
 anDropout)

 dense_40 (Dense)            (None, 256)               262400

 dropout_15 (Dropout)        (None, 256)               0

 dense_41 (Dense)            (None, 10)                2570

=================================================================
Total params: 1,632,650
Trainable params: 1,632,138
Non-trainable params: 512
```

Figure 20: CNN with BN and Pooling network design

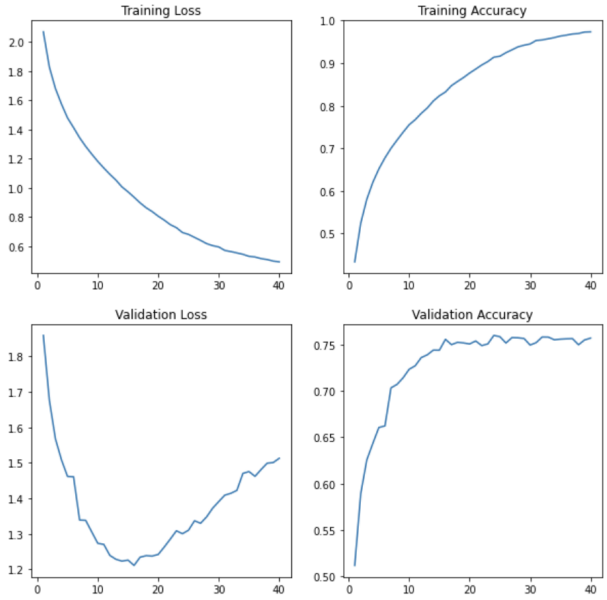


Figure 21: CNN with BN and Pooling result

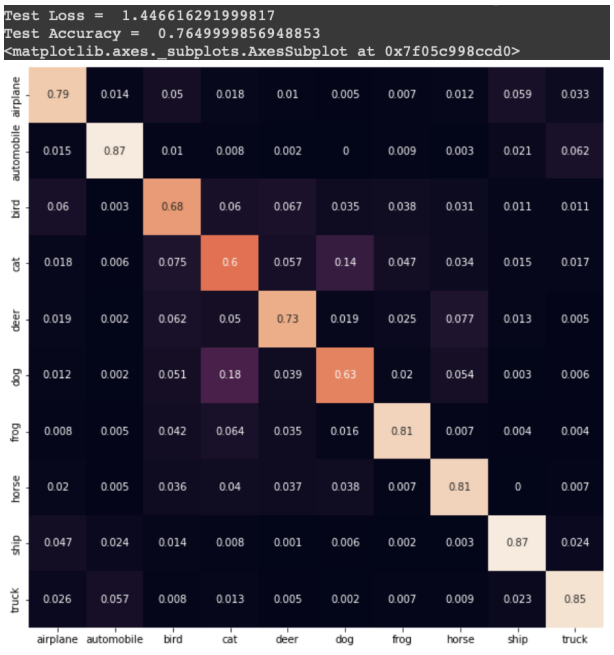


Figure 22: CNN with BN and Pooling result

2.10 Add Dropout

Dropout helps us to prevent network from overfitting. It randomly remove some connection by a specified rate for that layer and then train the network so the network do not just rely on specific connection. Our result exactly show it. Same validation accuracy as before and less train accuracy that shows preventing from over fitting.

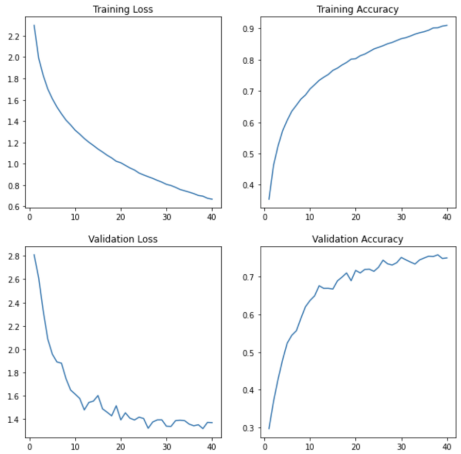


Figure 23: CNN with BN, Dropout, and Pooling result

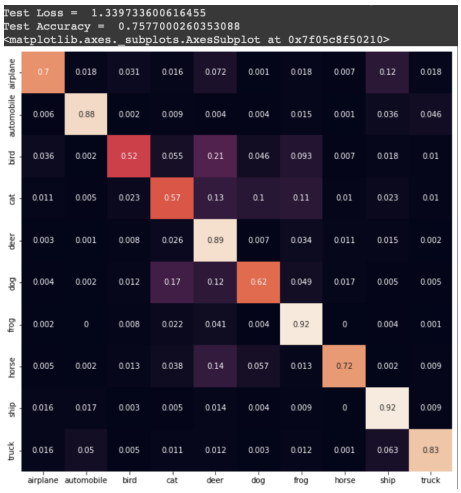


Figure 24: CNN with BN, Dropout, and Pooling result

3 Data Augmentation

3.1 What is data augmentation?

Data augmentation is a scenario in which we face lack of data in whole data set or specific category of data set so we make new data by rotation, scaling, flipping, or any other methods. The main point here is that by doing these transformations, the image label is same as before.

3.2 Augment sample image

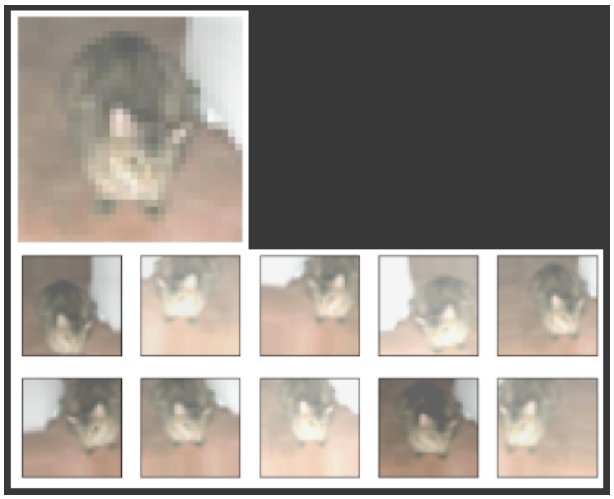
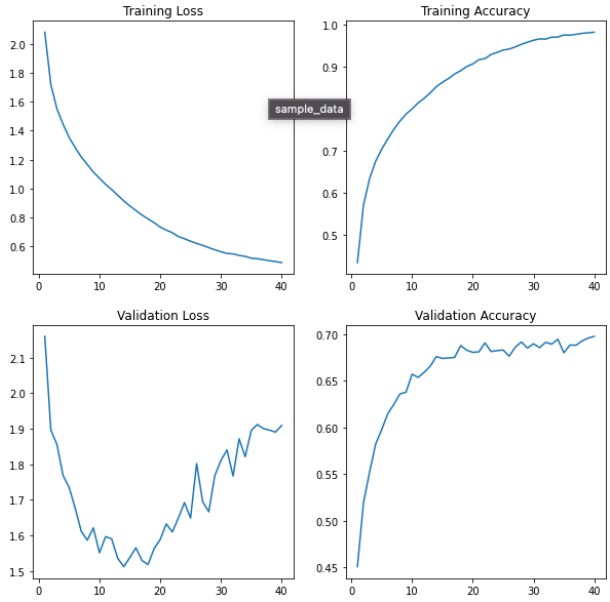


Figure 25: Augment sample image

3.3 Unbalanced data set

As we can see, network has almost random performance in dog and cat categories.



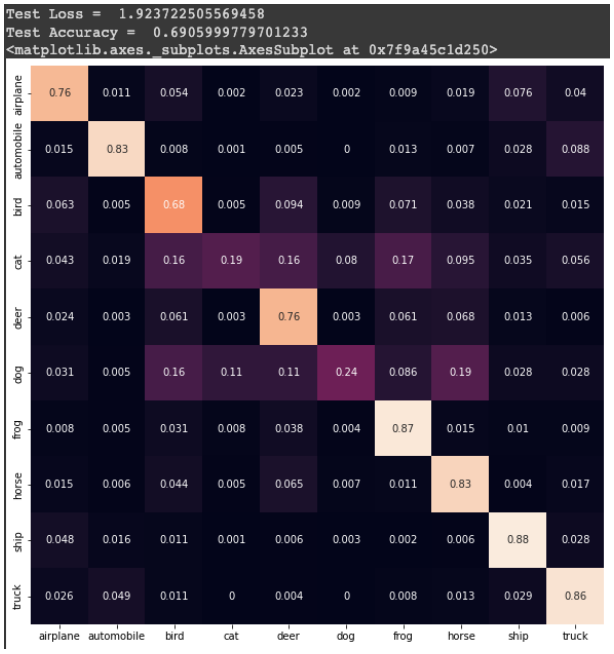
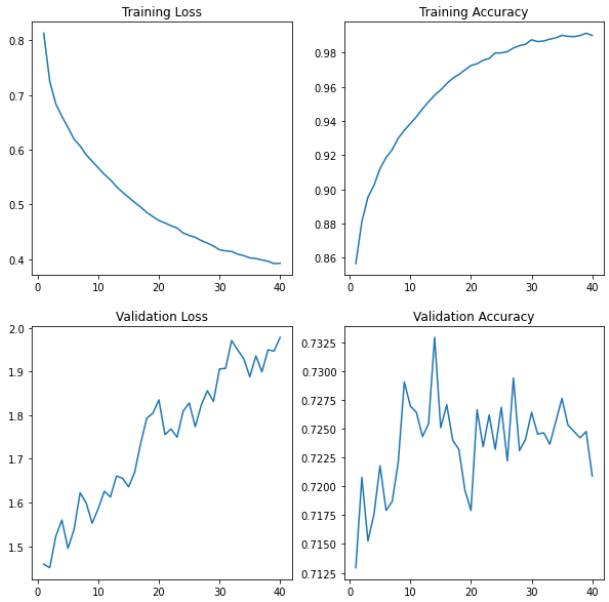
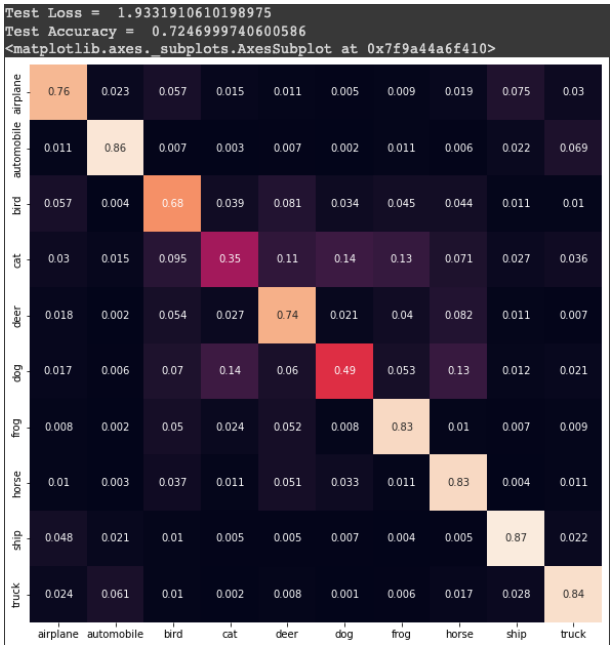


Figure 26: CNN with BN and Pooling result

3.4 Augment unbalanced data set

For solving this problem, I augmented each dog or cat image to make 9 new images for that categories. You can see that we have better performance in cat and dog categories but not as best as original data set!





4 Transfer learning (+)

4.1 VGG19 and SqueezeNet Architecture

Transfer Learning

VGG created by → Visual Geometry Group at Oxford

- Architecture :

2× Conv 3×3 (64) → Max Pool → 2× Conv 3×3 (128) → Max Pool → 4× Conv 3×3 (256) → ...

... → Max Pool → 4× Conv 3×3 (512) → Max Pool → 4× Conv 3×3 (512) → Max Pool → ...

... → Fully Connected (4096) → Fully Connected (4096) → Fully Connected (1000) → Soft Max

- Types : The architecture I explained above is model E.

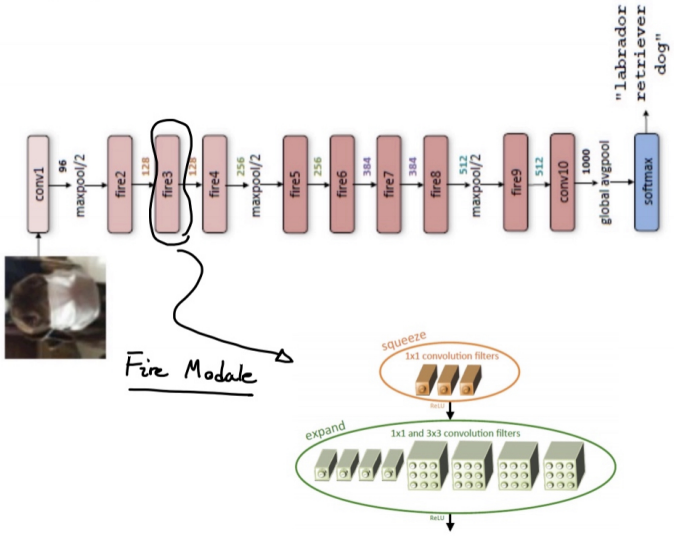
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

- Pros & Cons :
  - good classification architecture for many datasets
  - can be used for facial recognition
  - weights are easily available and can be transfered and reused
  - it has so many weight parameters so the model is very heavy 550 MB
  - long inference time

- Input & Output :
  - Input Image : fixed size ($224^{Px} \times 224^{Px}$) RGB image
  - Pre processing : subtracte the mean RGB value from each pixel over the whole training set
  - Output : 1000 channels classification

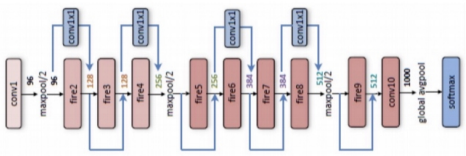
Squeeze Net ——→ 50 times less parameters than AlexNet but still maintains AlexNet level accuracy

- Architecture :

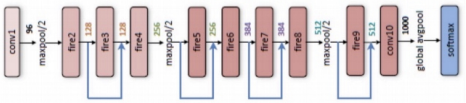


- Types : We can also add simple or complex bypass to the orginal SqueezeNet architecture.

Complex bypass :



Simple bypass :



SqueezeNet has also 8 bit & 6 bit types :

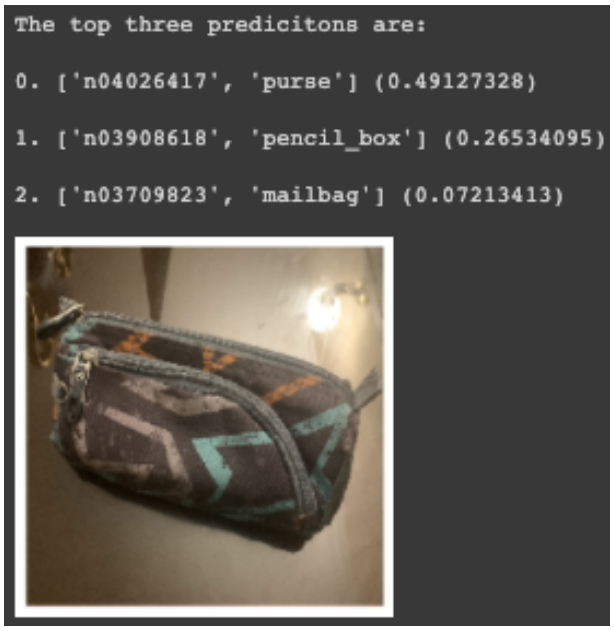
CNN architecture	Compression Approach	Data Type	Original → Compressed Model Size	Reduction in Model Size vs. AlexNet	Top-1 ImageNet Accuracy	Top-5 ImageNet Accuracy
AlexNet	None (baseline)	32 bit	240MB	1x	57.2%	80.3%
AlexNet	SVD (Denton et al., 2014)	32 bit	240MB → 48MB	5x	56.0%	79.4%
AlexNet	Network Pruning (Han et al., 2015b)	32 bit	240MB → 27MB	9x	57.2%	80.3%
AlexNet	Deep Compression (Han et al., 2015a)	5-8 bit	240MB → 6.9MB	35x	57.2%	80.3%
SqueezeNet (ours)	None	32 bit	4.8MB	**50x**	57.5%	80.3%
SqueezeNet (ours)	Deep Compression	8 bit	4.8MB → 0.66MB	**363x**	57.5%	80.3%
SqueezeNet (ours)	Deep Compression	6 bit	4.8MB → 0.47MB	**510x**	57.5%	80.3%

- Pros & Cons :
  - 50 times less parameter than AlexNet
  - AlexNet accuracy on ImageNet dataset
  - Using deep compression, we can fit this model on devices such as FPGAs

```
Model: "vgg19"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
block3_conv4 (Conv2D)        (None, 56, 56, 256)       590080
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
block4_conv4 (Conv2D)        (None, 28, 28, 512)       2359808
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
block5_conv4 (Conv2D)        (None, 14, 14, 512)       2359808
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
flatten (Flatten)            (None, 25088)             0
fc1 (Dense)                  (None, 4096)              102764544
fc2 (Dense)                  (None, 4096)              16781312
predictions (Dense)          (None, 1000)              4097000
=================================================================
Total params: 143,667,240
```

Figure 27: VGG19 network

4.2 Test VGG19 on sample image



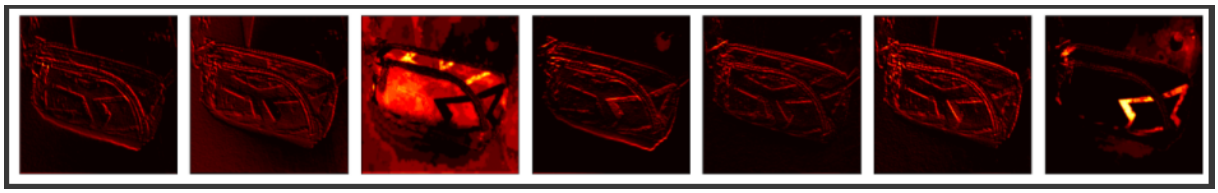


Figure 28: First Layer Feature Map

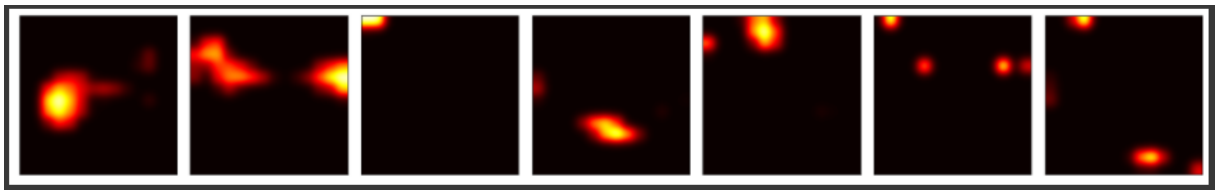


Figure 29: Last Layer Feature Map

4.3 What can we do with images out of the data set categories?

I described this part in the VGG19 notebook in this github reposiroty with some sample images.

4.4 Retrain VGG19 on two category dataset

I did this part in the VGG19 notebook in this github reposiroty.

5 Pose Estimation (+)

I did this part in the pose-estimation notebook in this github reposiroty. The data set contains only 2000 samples and is really small even if you do data augmentation. As I read, other codes have more complex networks and using extended data sets.

Figure 30: Joints data

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 55, 55, 96)        34944

 lambda (Lambda)             (None, 55, 55, 96)        0

 max_pooling2d (MaxPooling2D  (None, 27, 27, 96)       0
 )

 conv2d_1 (Conv2D)           (None, 27, 27, 256)       614656

 lambda_1 (Lambda)           (None, 27, 27, 256)       0

 max_pooling2d_1 (MaxPooling  (None, 13, 13, 256)      0
 2D)

 conv2d_2 (Conv2D)           (None, 13, 13, 384)       885120

 conv2d_3 (Conv2D)           (None, 13, 13, 384)       1327488

 conv2d_4 (Conv2D)           (None, 13, 13, 256)       884992

 max_pooling2d_2 (MaxPooling  (None, 6, 6, 256)        0
 2D)

 flatten (Flatten)           (None, 9216)              0

 dense (Dense)               (None, 4096)              37752832

 dropout (Dropout)           (None, 4096)              0

 dense_1 (Dense)             (None, 4096)              16781312

 dropout_1 (Dropout)         (None, 4096)              0

 dense_2 (Dense)             (None, 28)                114716

=================================================================
Total params: 58,396,060
Trainable params: 58,396,060
Non-trainable params: 0
_________________________________________________________________
```

Figure 31: Implementation of the network described in the paper