Deep Learning: Homework #3

Due on December 3, 2019 at 11:55pm

Professor Emad Fatemizadeh

Amirhossein Yousefi

97206984

Practical Excercise 1

Part2:

Here is some sample of *Cifar10* in bellow figure.

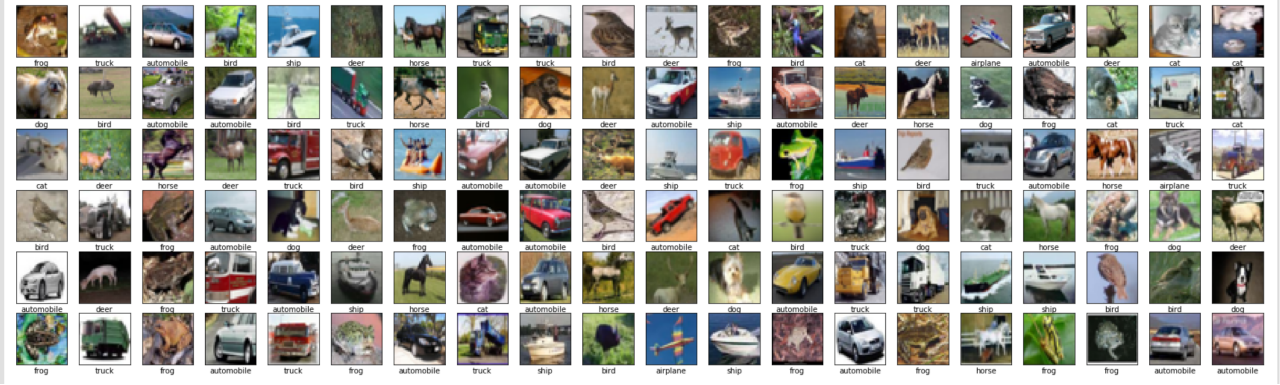


Figure 1: Cifar10

Part4:
Normalization makes dynamic range similar and stabilize weights of neural network and facilitate the process of training.Here it's better to divide samples by 255 in order to assimilate the range of samples.

Part6:
The architecture of the network is show in bellow figure and learning rate is 10^{-3} , batch size is128 and optimizer is *Adam*.Here we used keras as API and keras itself provide us with accuracy and it makes data into arbitrary batches.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 30, 30, 32)        896
_________________________________________________________________
batch_normalization (BatchNo (None, 30, 30, 32)        128
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 13, 13, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 4, 4, 128)         73856
_________________________________________________________________
dropout (Dropout)            (None, 4, 4, 128)         0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 2, 2, 128)         147584
_________________________________________________________________
flatten (Flatten)            (None, 512)               0
_________________________________________________________________
dense (Dense)                (None, 128)               65664
_________________________________________________________________
batch_normalization_1 (Batch (None, 128)               512
_________________________________________________________________
dense_1 (Dense)              (None, 128)               16512
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290
=================================================================
Total params: 324,938
Trainable params: 324,618
Non-trainable params: 320
_________________________________________________________________
```

Figure 2: Network structure

Part7:
In bellow figure the process of training is depicted.

```
  128/45000 [..............................] - ETA: 22:54 - loss: 2.4661 - acc: 0.1094WARNING:tensorflow:Method (on_train_batch_end) is s
45000/45000 [==============================] - 26s 581us/sample - loss: 1.5691 - acc: 0.4218 - val_loss: 1.5734 - val_acc: 0.4574
Epoch 2/20
45000/45000 [==============================] - 21s 478us/sample - loss: 1.1513 - acc: 0.5857 - val_loss: 1.2997 - val_acc: 0.5570
Epoch 3/20
45000/45000 [==============================] - 21s 475us/sample - loss: 0.9959 - acc: 0.6441 - val_loss: 1.0566 - val_acc: 0.6276
Epoch 4/20
45000/45000 [==============================] - 21s 472us/sample - loss: 0.9017 - acc: 0.6816 - val_loss: 0.9517 - val_acc: 0.6612
Epoch 5/20
45000/45000 [==============================] - 21s 469us/sample - loss: 0.8309 - acc: 0.7088 - val_loss: 0.9985 - val_acc: 0.6562
Epoch 6/20
45000/45000 [==============================] - 21s 473us/sample - loss: 0.7753 - acc: 0.7278 - val_loss: 0.8774 - val_acc: 0.6994
Epoch 7/20
45000/45000 [==============================] - 21s 467us/sample - loss: 0.7257 - acc: 0.7460 - val_loss: 0.7731 - val_acc: 0.7330
Epoch 8/20
45000/45000 [==============================] - 21s 469us/sample - loss: 0.6899 - acc: 0.7564 - val_loss: 1.2613 - val_acc: 0.6220
Epoch 9/20
45000/45000 [==============================] - 21s 468us/sample - loss: 0.6605 - acc: 0.7688 - val_loss: 0.7696 - val_acc: 0.7384
Epoch 10/20
45000/45000 [==============================] - 21s 468us/sample - loss: 0.6247 - acc: 0.7797 - val_loss: 0.7959 - val_acc: 0.7256
Epoch 11/20
45000/45000 [==============================] - 21s 462us/sample - loss: 0.5920 - acc: 0.7929 - val_loss: 0.7652 - val_acc: 0.7446
Epoch 12/20
45000/45000 [==============================] - 21s 469us/sample - loss: 0.5678 - acc: 0.7997 - val_loss: 0.8496 - val_acc: 0.7148
Epoch 13/20
45000/45000 [==============================] - 21s 471us/sample - loss: 0.5452 - acc: 0.8093 - val_loss: 0.9319 - val_acc: 0.7000
Epoch 14/20
45000/45000 [==============================] - 21s 462us/sample - loss: 0.5225 - acc: 0.8159 - val_loss: 0.7971 - val_acc: 0.7470
Epoch 15/20
45000/45000 [==============================] - 21s 463us/sample - loss: 0.4964 - acc: 0.8248 - val_loss: 0.7914 - val_acc: 0.7486
Epoch 16/20
45000/45000 [==============================] - 21s 468us/sample - loss: 0.4845 - acc: 0.8292 - val_loss: 0.8022 - val_acc: 0.7440
Epoch 17/20
45000/45000 [==============================] - 21s 472us/sample - loss: 0.4662 - acc: 0.8330 - val_loss: 0.6946 - val_acc: 0.7806
Epoch 18/20
45000/45000 [==============================] - 21s 467us/sample - loss: 0.4442 - acc: 0.8430 - val_loss: 0.6604 - val_acc: 0.7844
Epoch 19/20
45000/45000 [==============================] - 21s 471us/sample - loss: 0.4305 - acc: 0.8491 - val_loss: 0.7359 - val_acc: 0.7776
Epoch 20/20
45000/45000 [==============================] - 21s 460us/sample - loss: 0.4175 - acc: 0.8520 - val_loss: 0.6954 - val_acc: 0.7754
```

Figure 3: Training process

Here is accuracy on test set and also accuracy on train and validation set in each epoch .Accuracy calculation is done by predefined functions in *keras*.

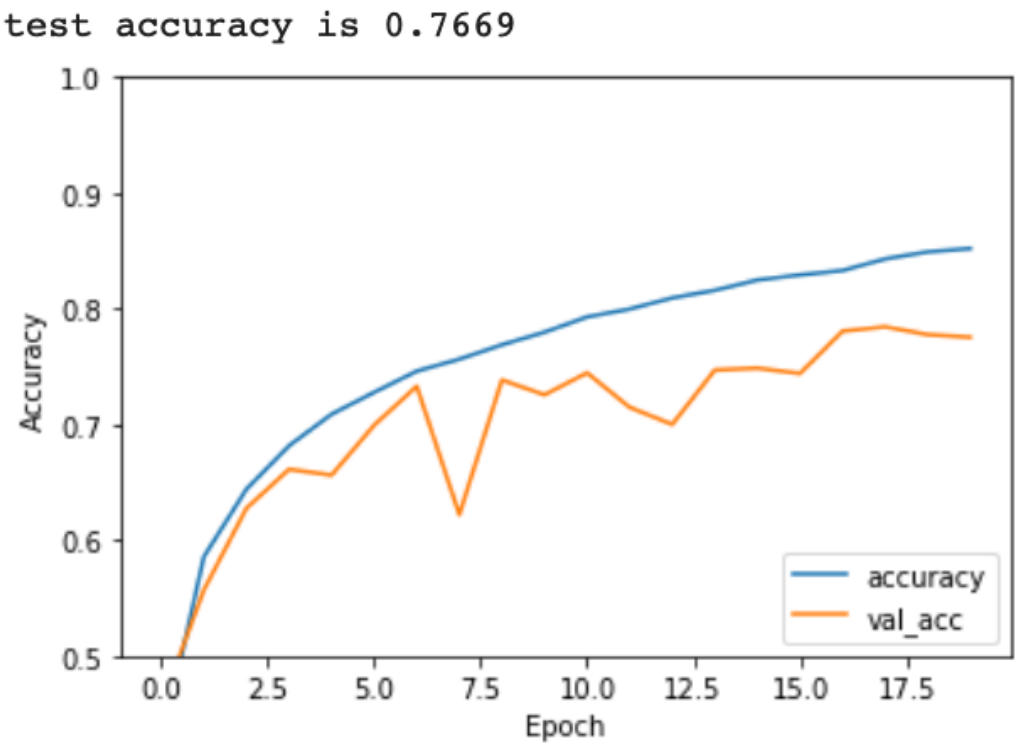


Figure 4: Accuracy

Part8:
Tensorboard visualization is done in *Callback* in *Keras*.Here is visualization.

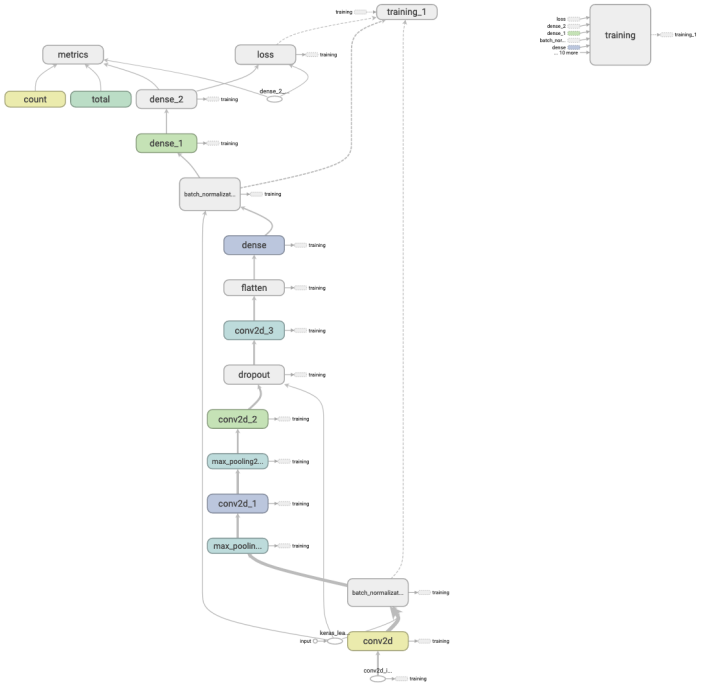
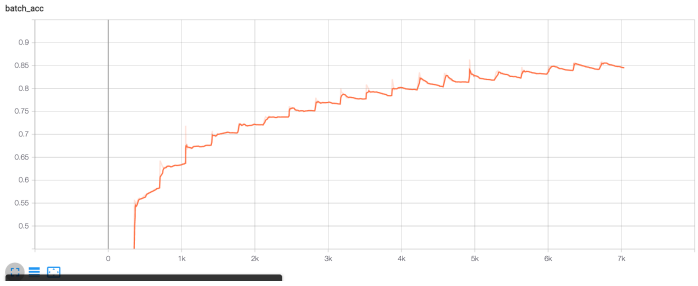
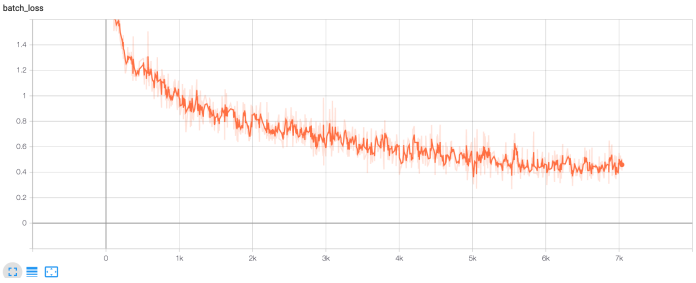


Figure 5: Computation graph

Batch accuracy and loss are measured in one complete epoch.



(a) Accuracy

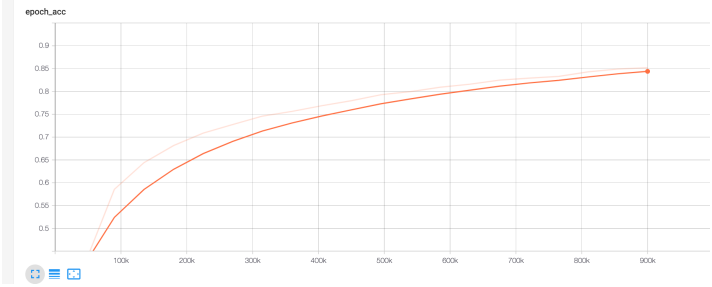


(b) Loss

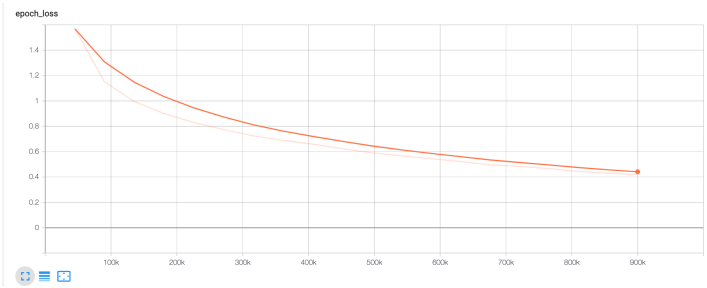
Figure 6: Batch results(train set)

Problem 1 continued on next page. . .

Accuracy and loss also is measured in every batch size in whole 20 epochs.



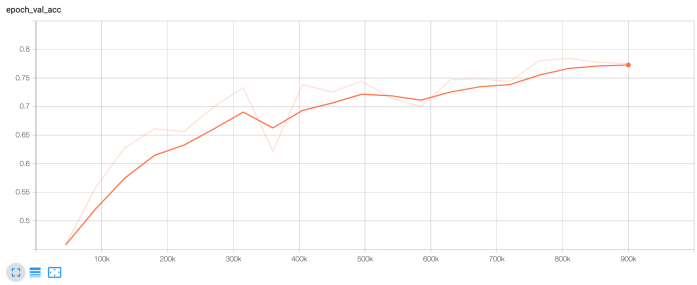
(a) Accuracy



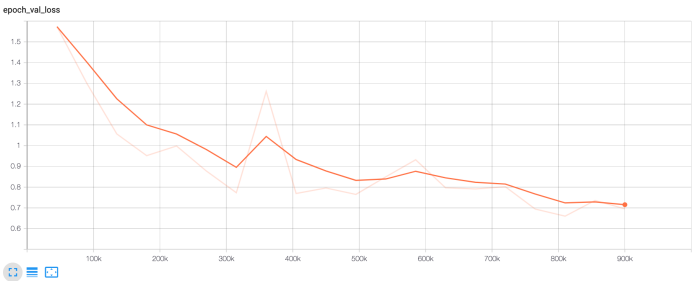
(b) Loss

Figure 7: Epoch results(train set)

Next loss and accuracy is calculated on validation set.



(a) Accuracy

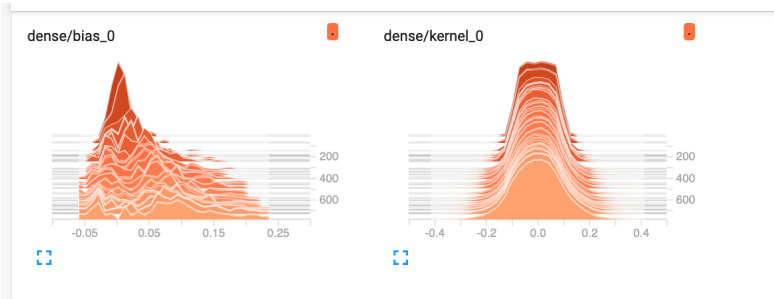


(b) Loss

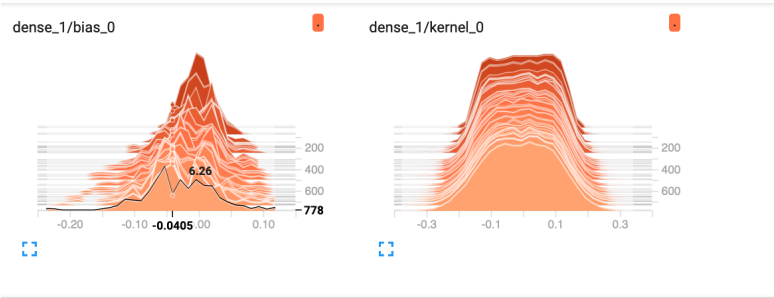
Figure 8: Epoch results(validation set)

Now let's visualize *histogram* of weights.

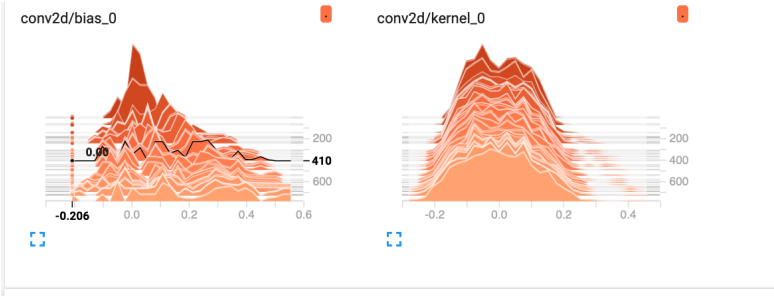
(a) Histogram(Batch normalization layer)



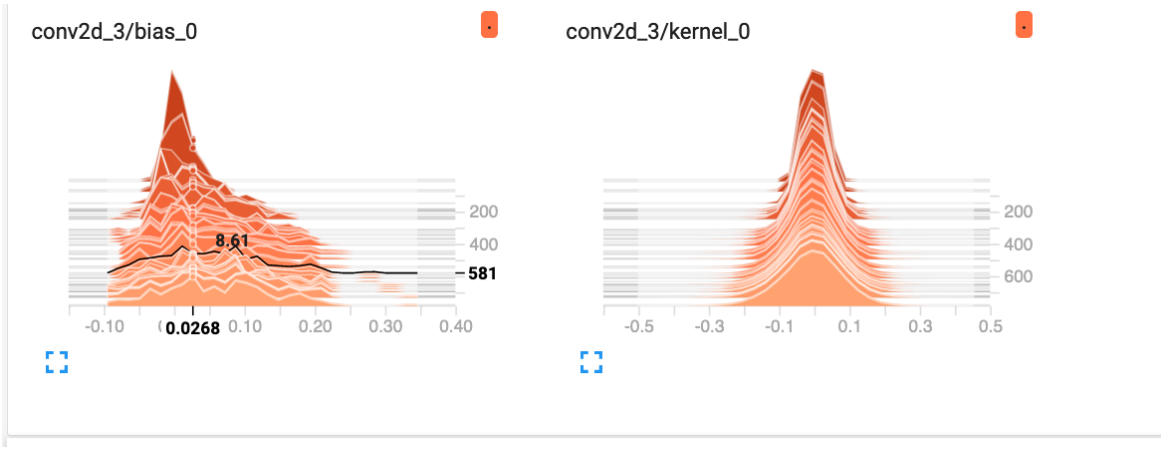
(a) Histogram(Dense layer)



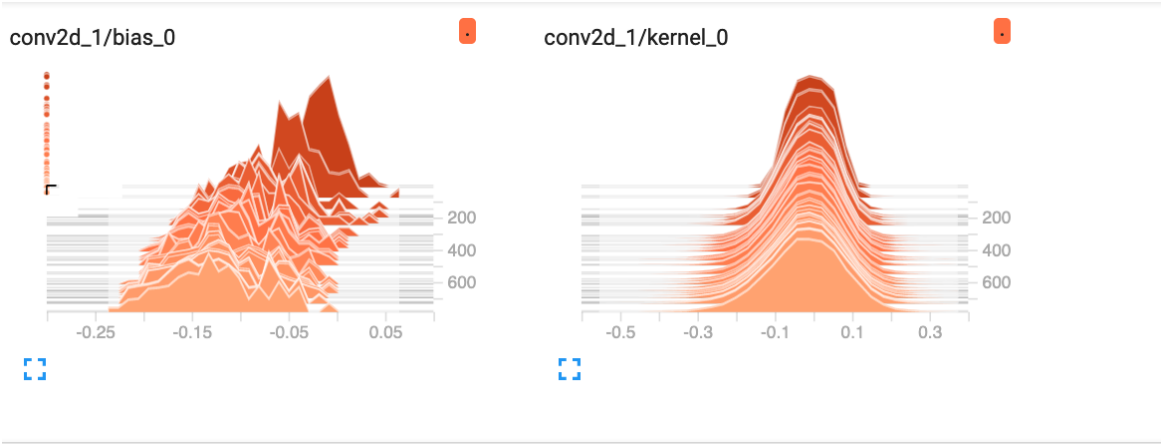
(a) Histogram(Dense layer)



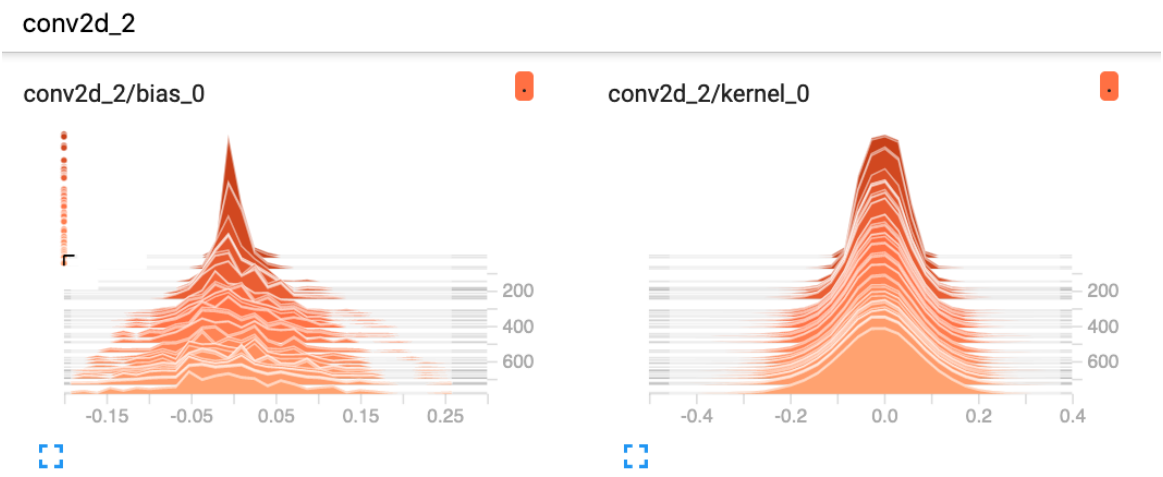
(a) Histogram(Convolution layer)



(a) Histogram(Convolution layer)



(a) Histogram(Convolution layer)



(a) Histogram(Convolution layer)

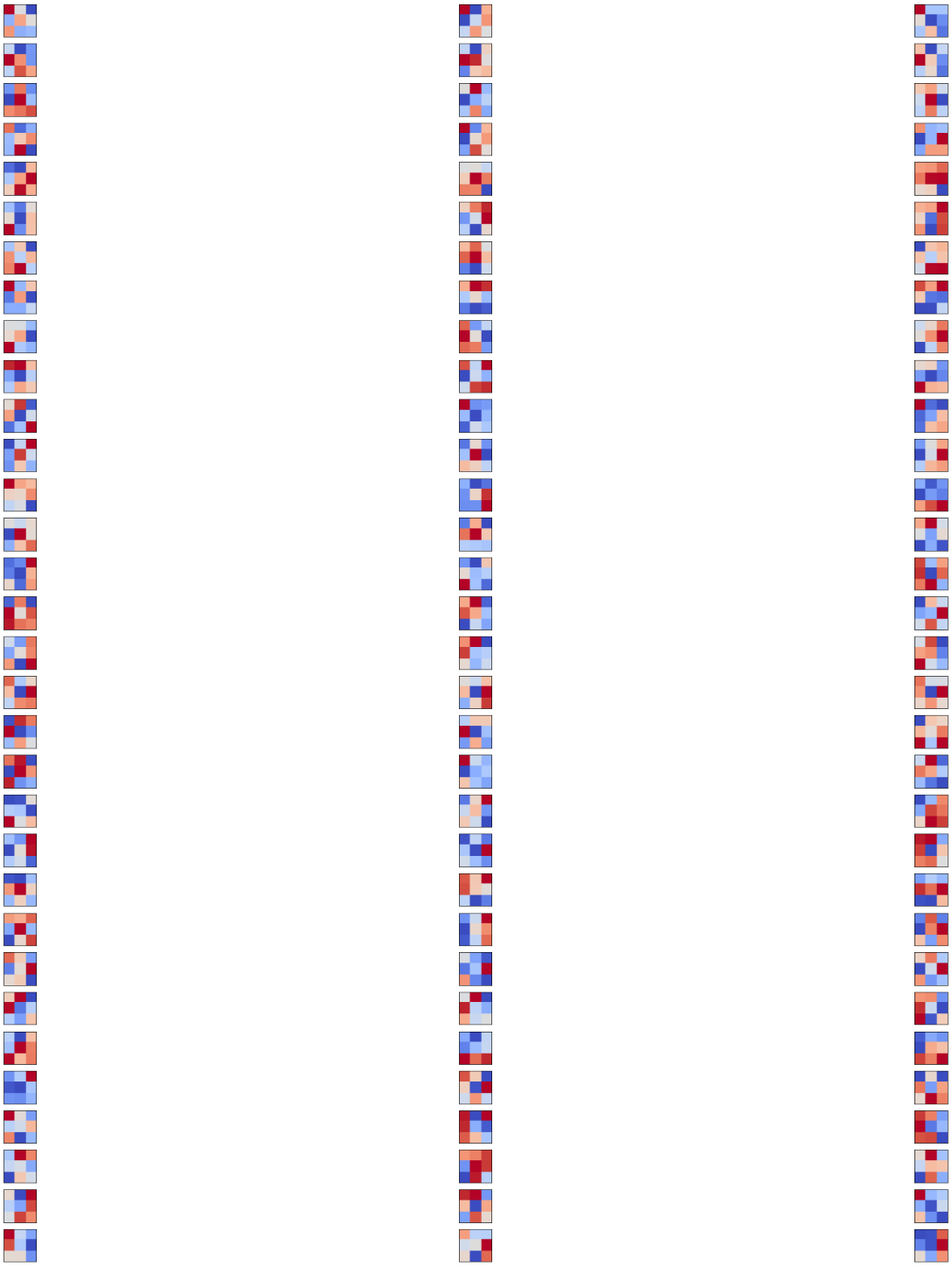
Problem 1 continued on next page. . .

Part9:

Test accuracy is 76.69 % according to top of figure4.

Part10:

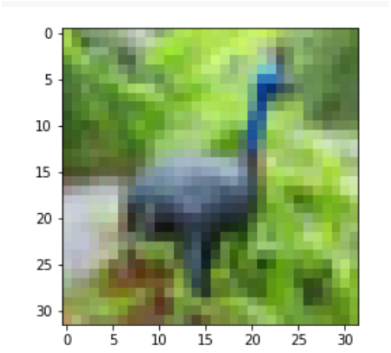
Now weights of first layer of trained networks is shown in bellow figure.Each kernel extracts particular pattern of input like,edge and intensity.



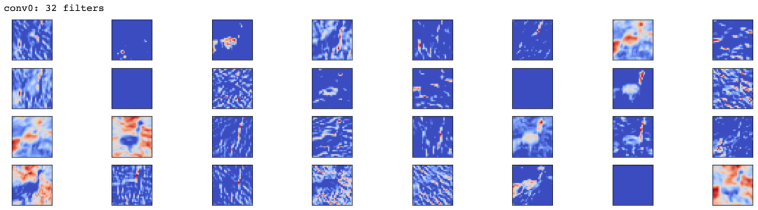
(a) Visualization of weights of first conv layer

Part11:

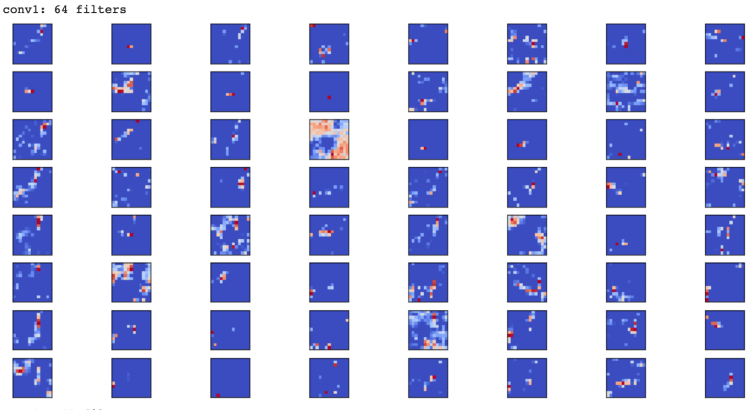
Now output of filters is shown in bellow figure for each sample.



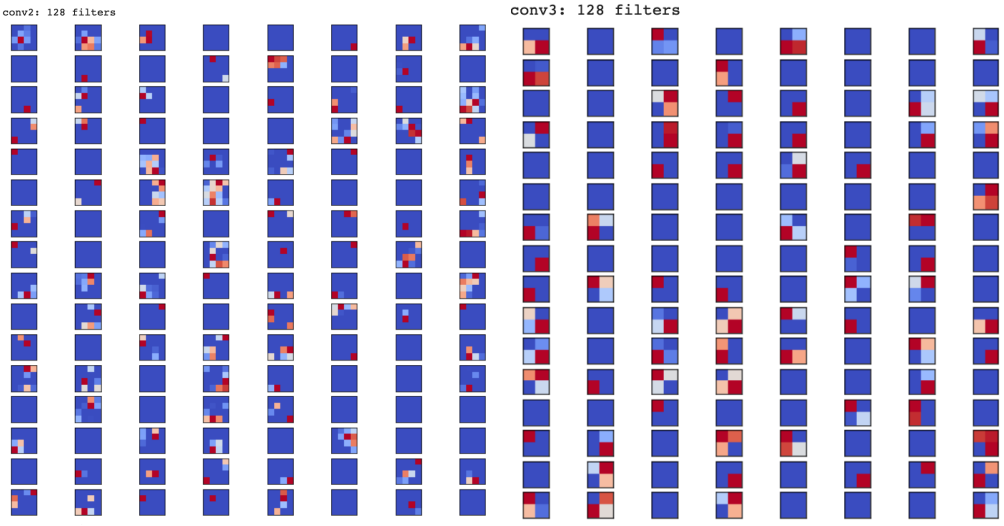
(a) Photo



(b) First conv output

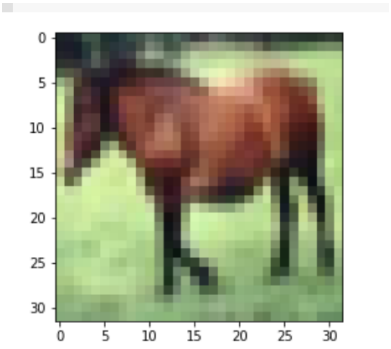


(c) Second conv output

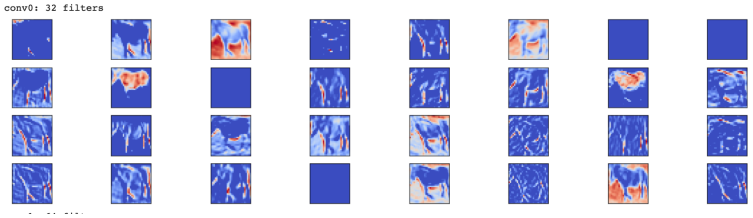


(d) Third conv output

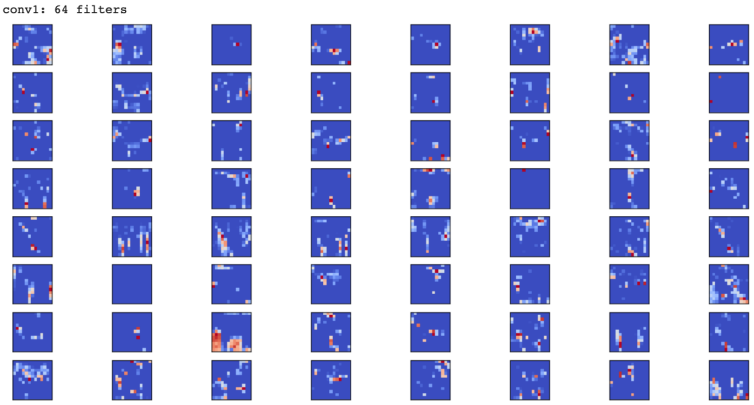
(e) Fourth conv output



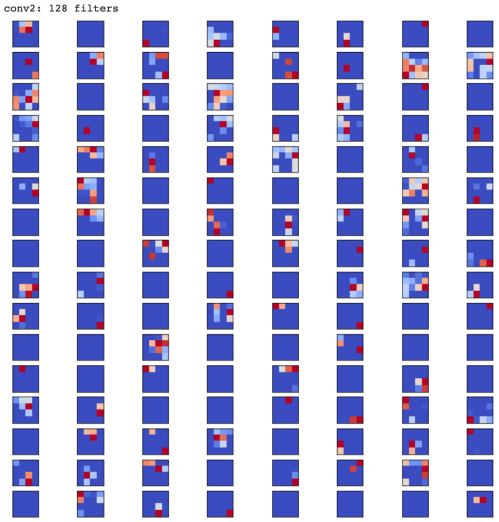
(a) Photo



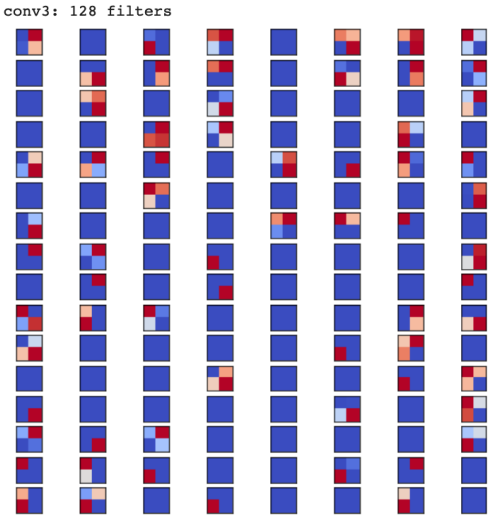
(b) First conv output



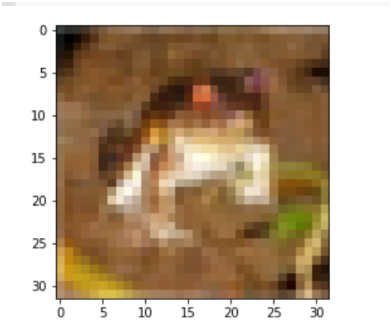
(c) Second conv output



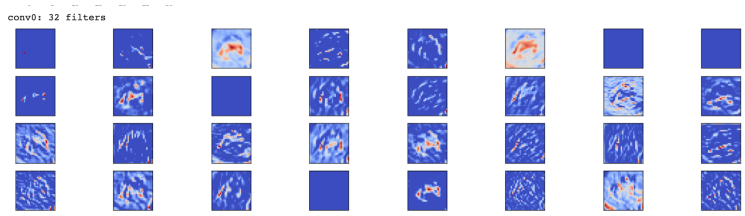
(d) Third conv output



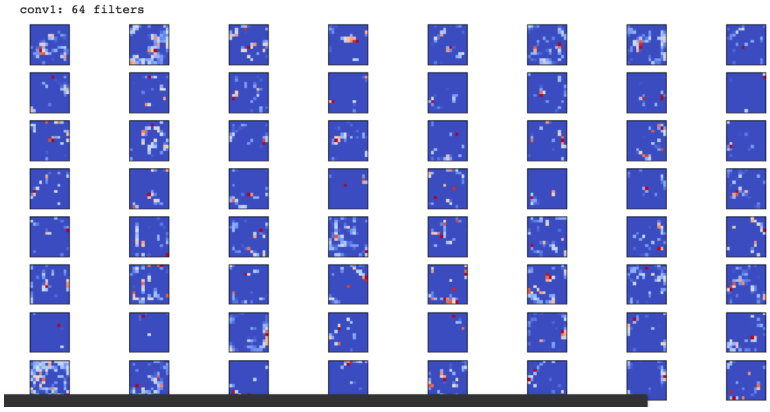
(e) Fourth conv output



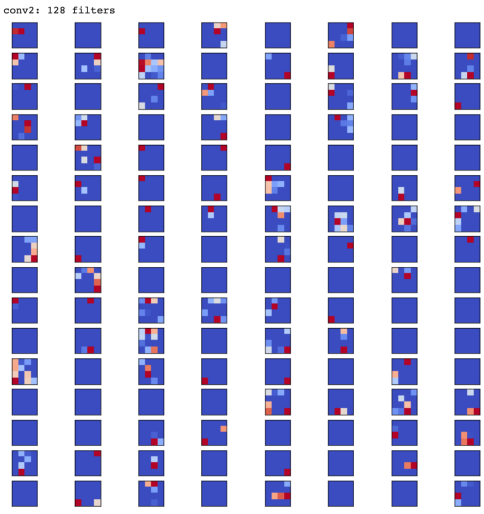
(a) Photo



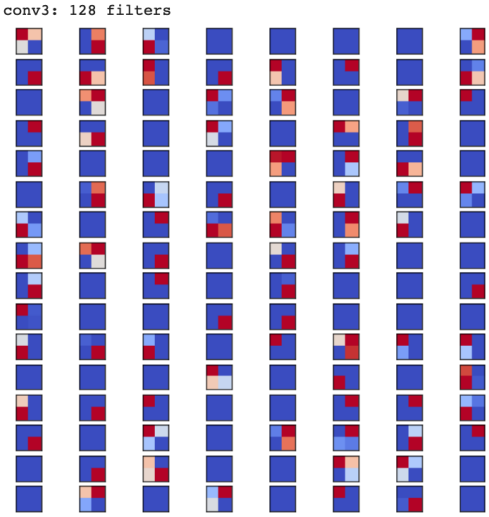
(b) First conv output



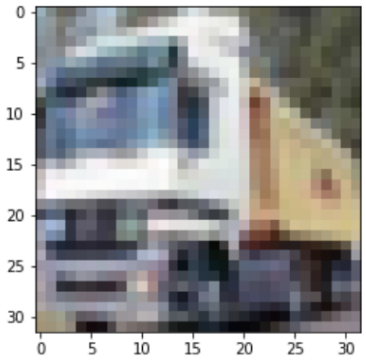
(c) Second conv output



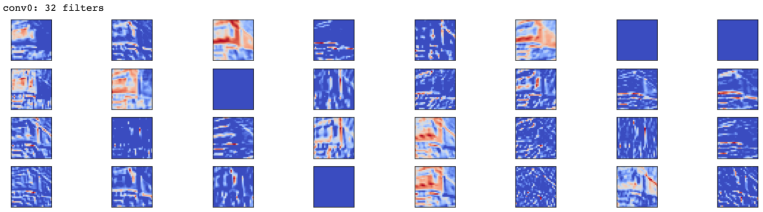
(d) Third conv output



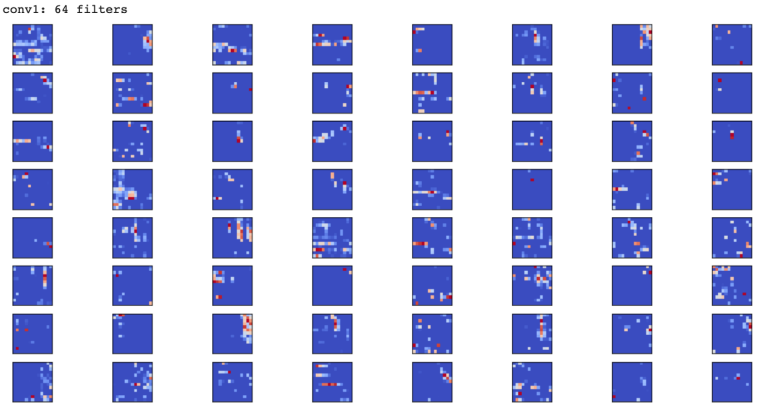
(e) Fourth conv output



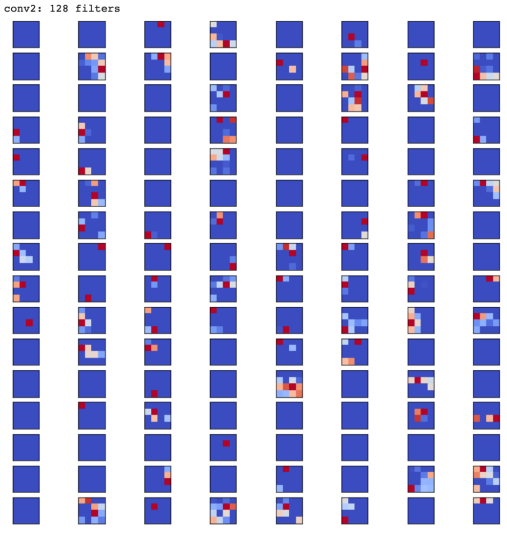
(a) Photo



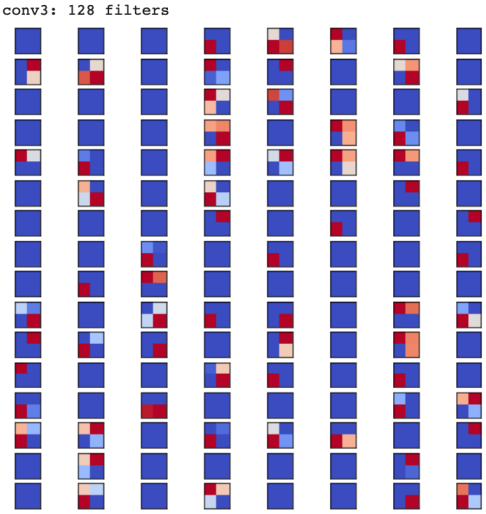
(b) First conv output



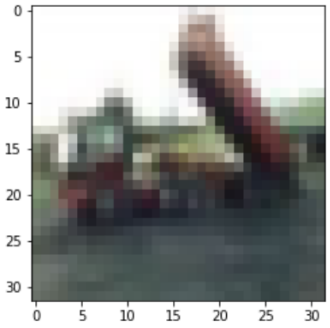
(c) Second conv output



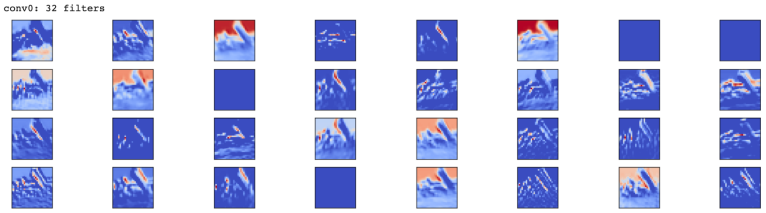
(d) Third conv output



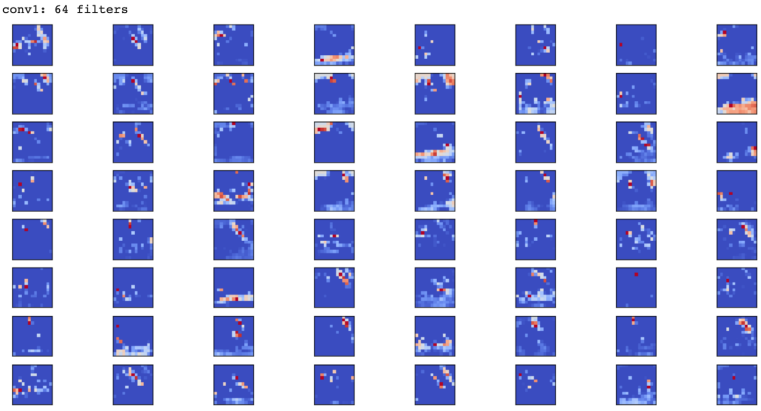
(e) Fourth conv output



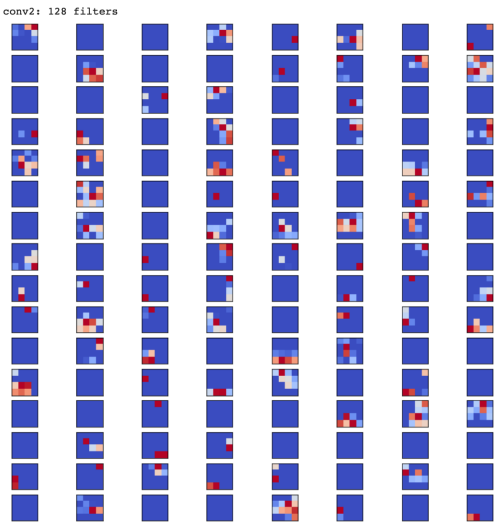
(a) Photo



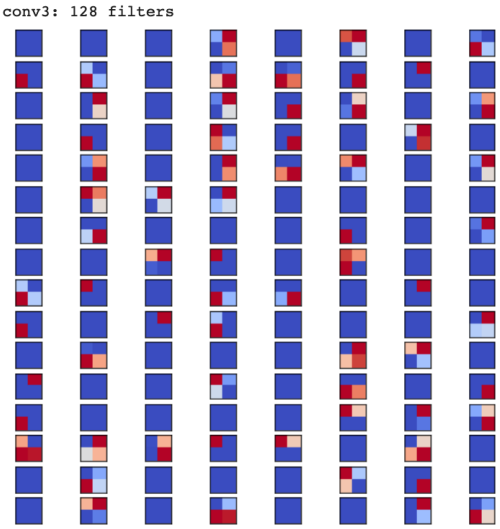
(b) First conv output



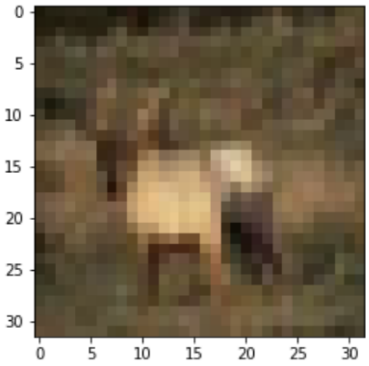
(c) Second conv output



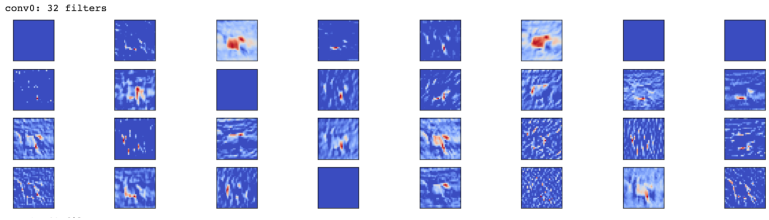
(d) Third conv output



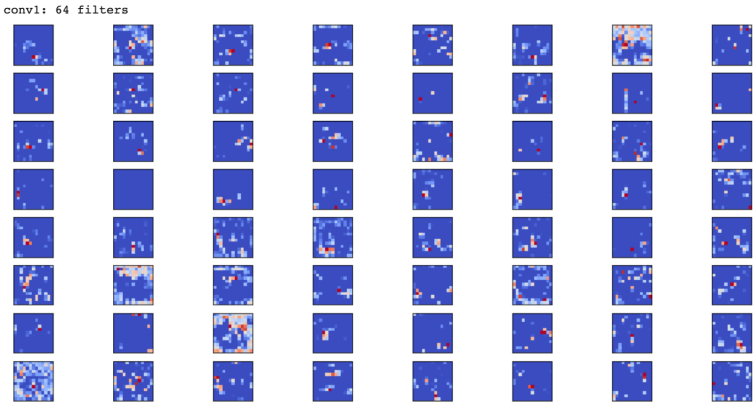
(e) Fourth conv output



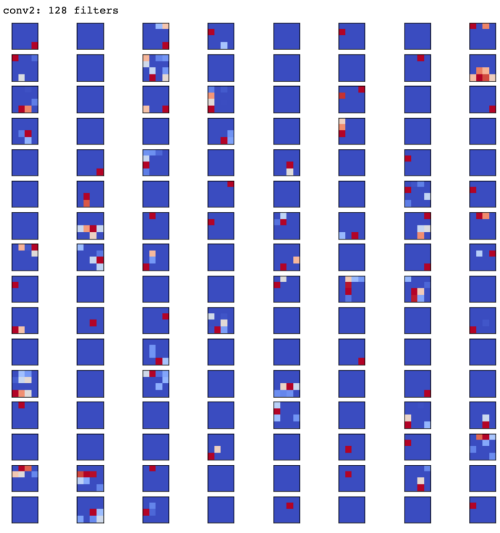
(a) Photo



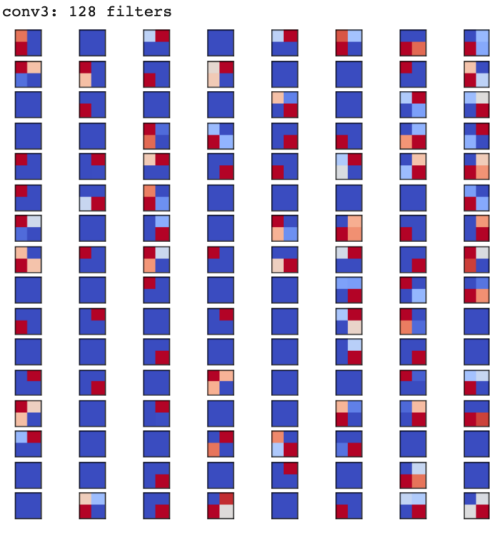
(b) First conv output



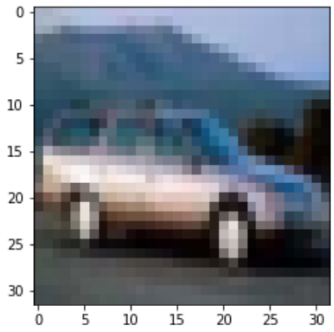
(c) Second conv output



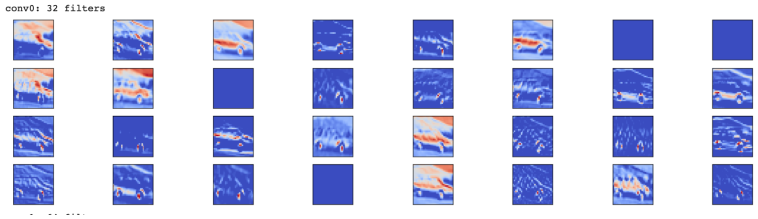
(d) Third conv output



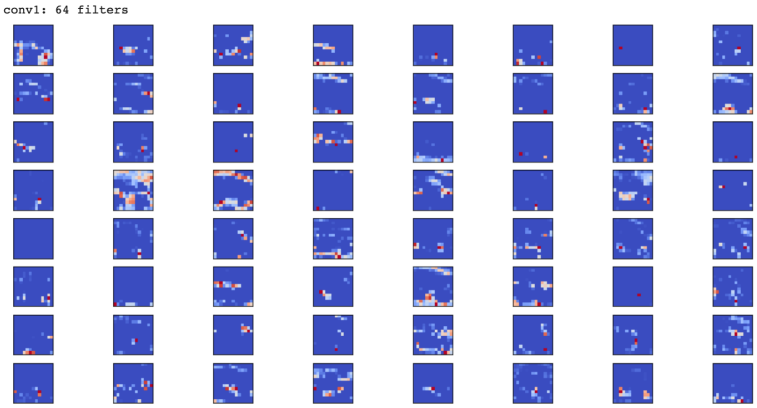
(e) Fourth conv output



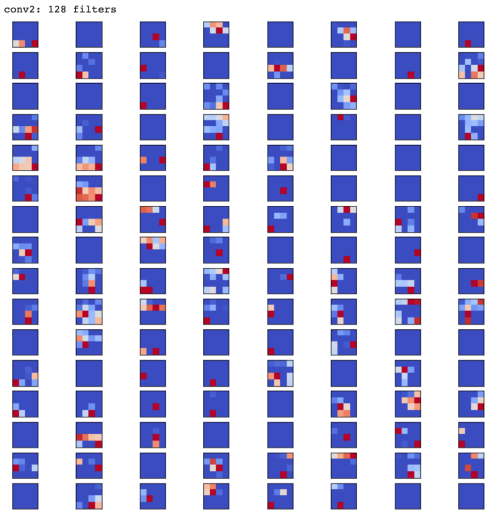
(a) Photo



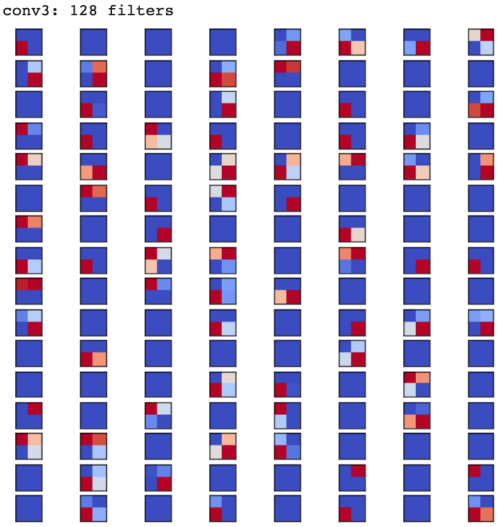
(b) First conv output



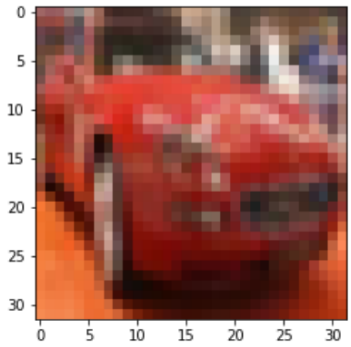
(c) Second conv output



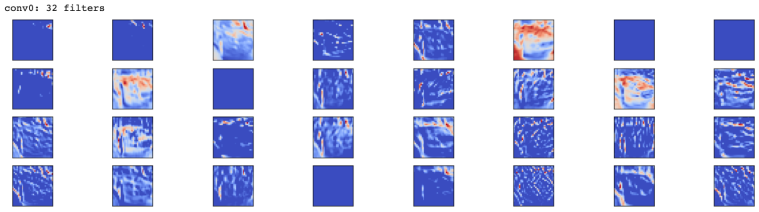
(d) Third conv output



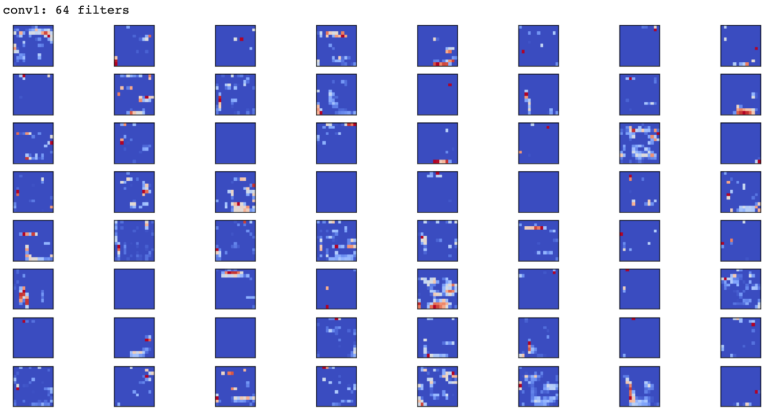
(e) Fourth conv output



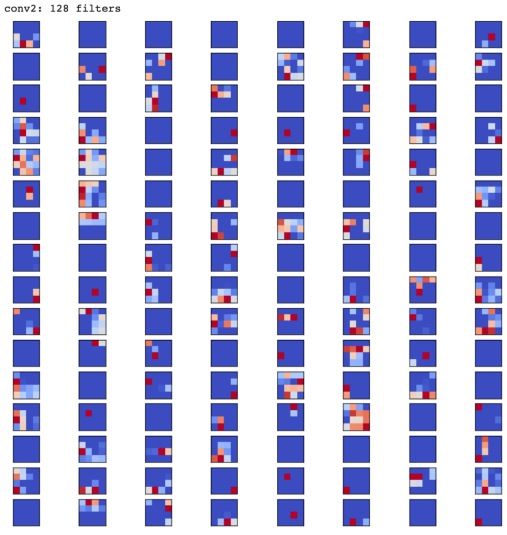
(a) Photo



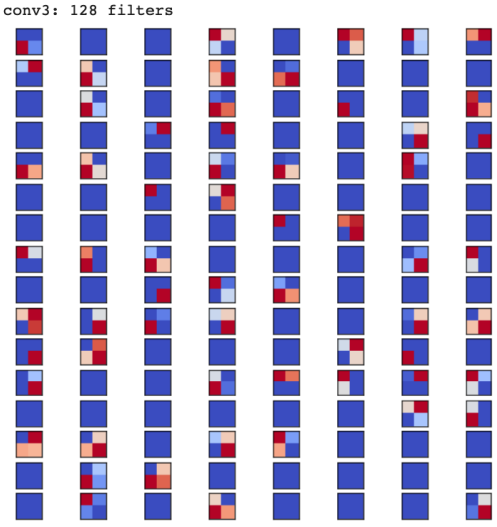
(b) First conv output



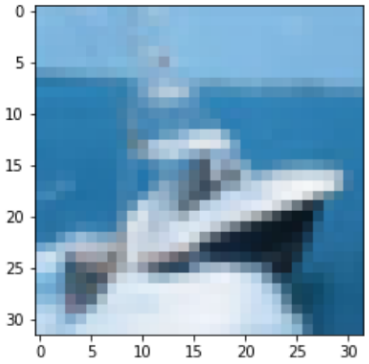
(c) Second conv output



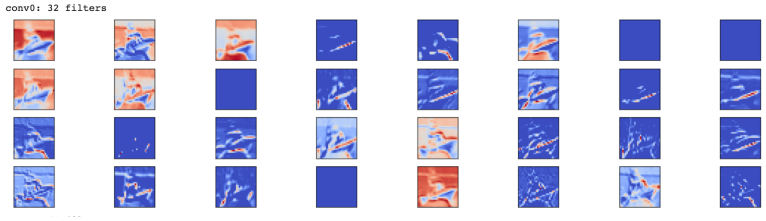
(d) Third conv output



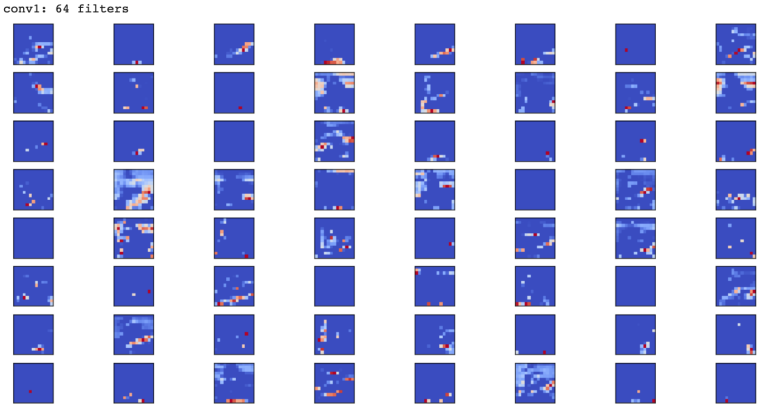
(e) Fourth conv output



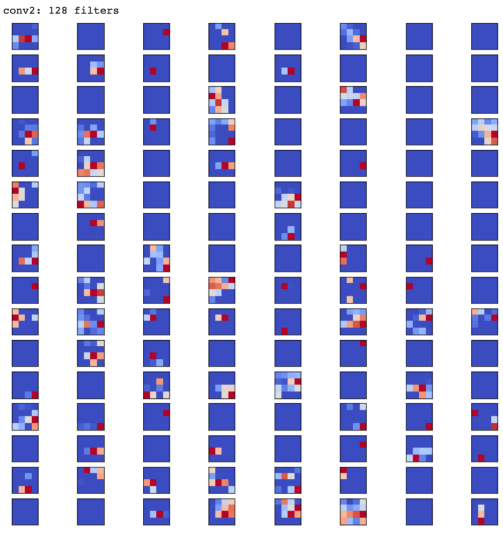
(a) Photo



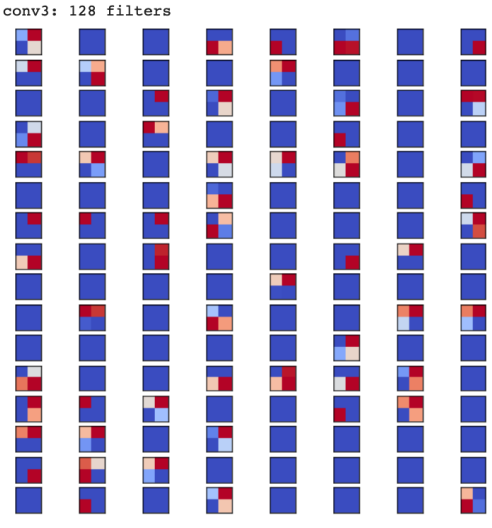
(b) First conv output



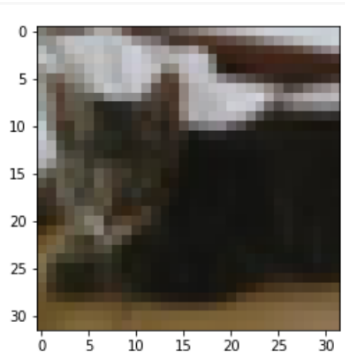
(c) Second conv output



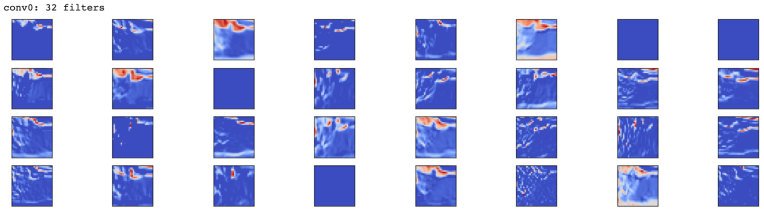
(d) Third conv output



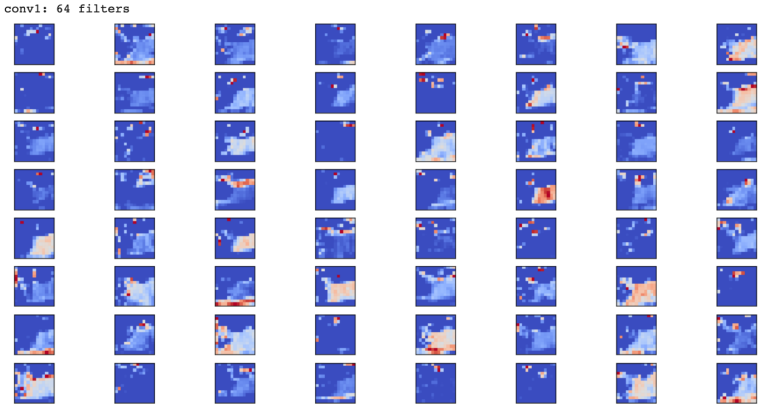
(e) Fourth conv output



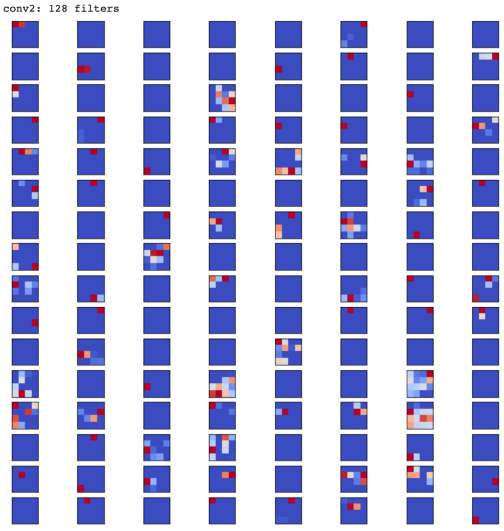
(a) Photo



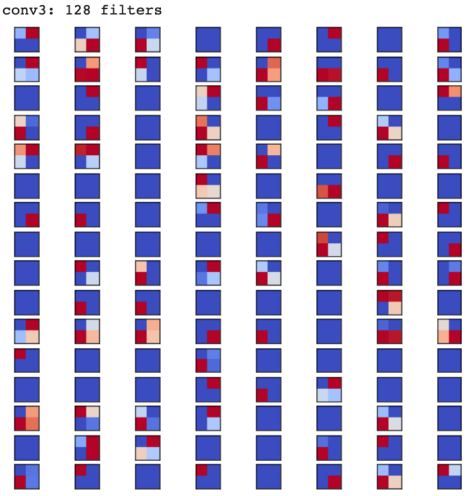
(b) First conv output



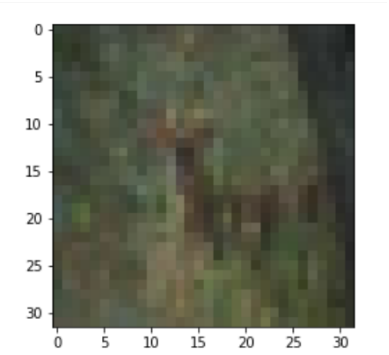
(c) Second conv output



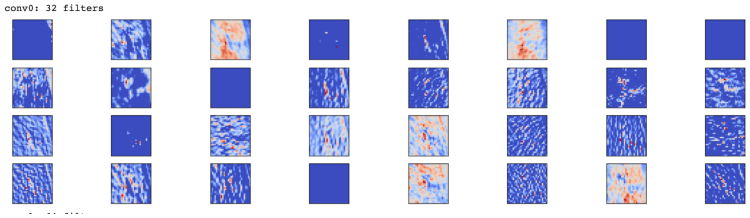
(d) Third conv output



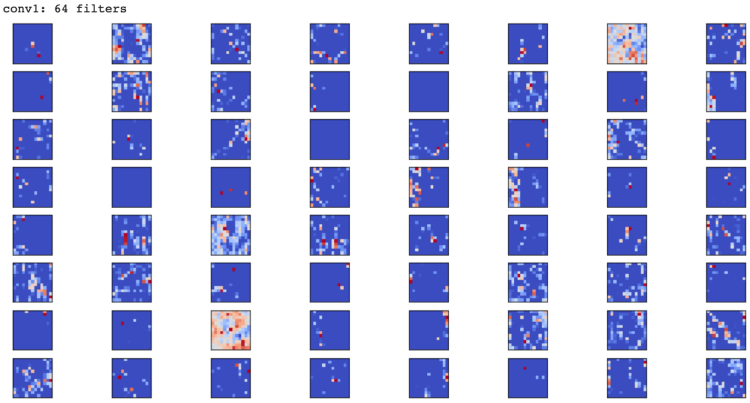
(e) Fourth conv output



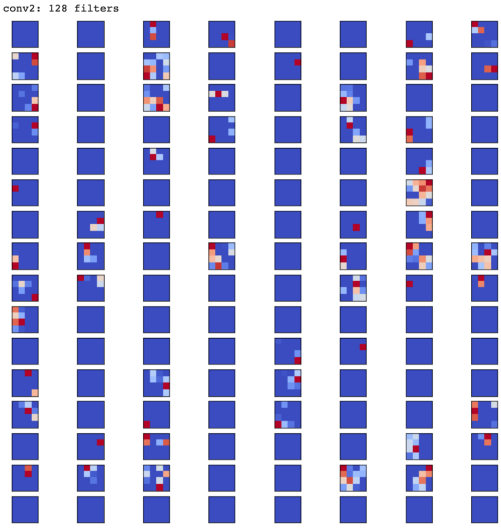
(a) Photo



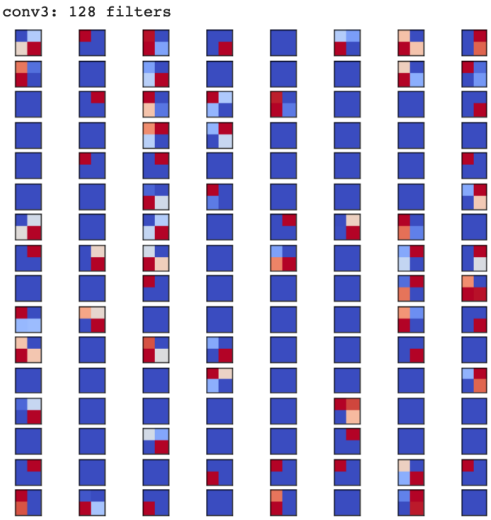
(b) First conv output



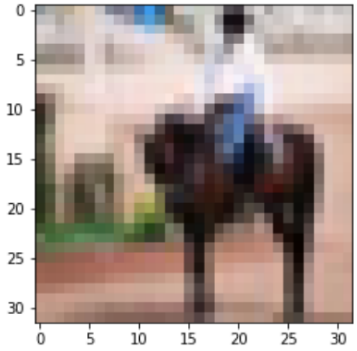
(c) Second conv output



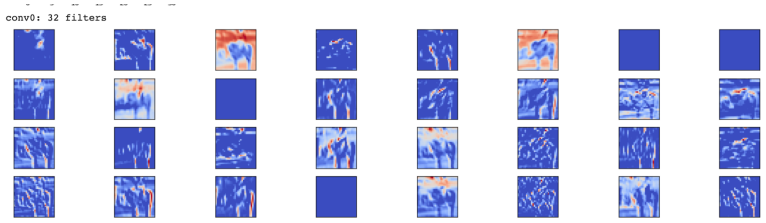
(d) Third conv output



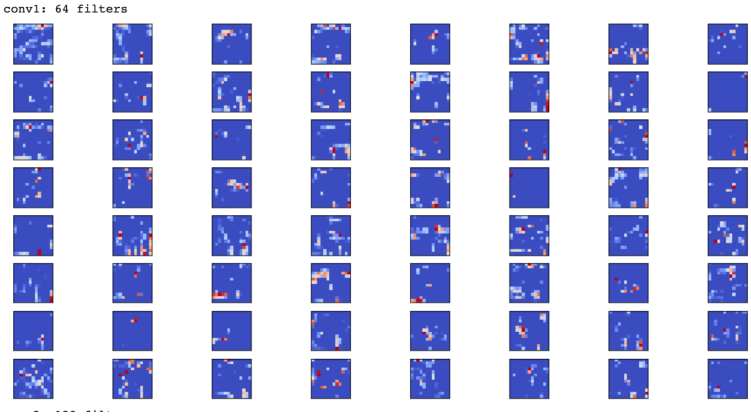
(e) Fourth conv output



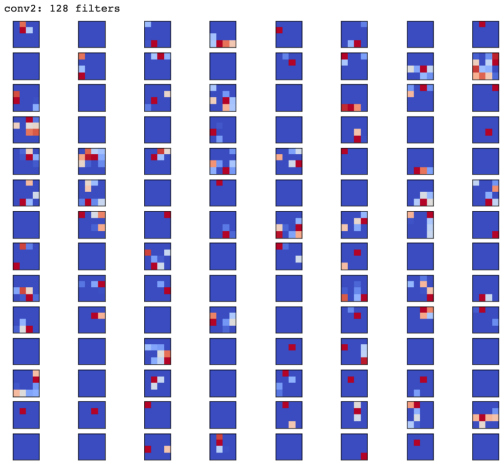
(a) Photo



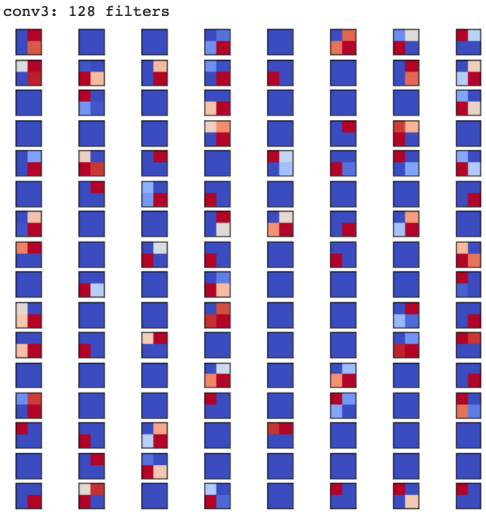
(b) First conv output



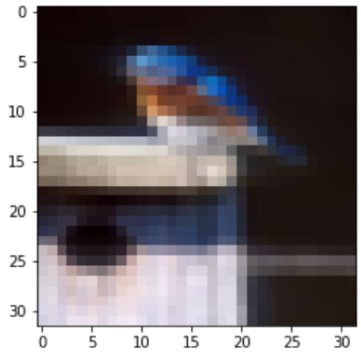
(c) Second conv output



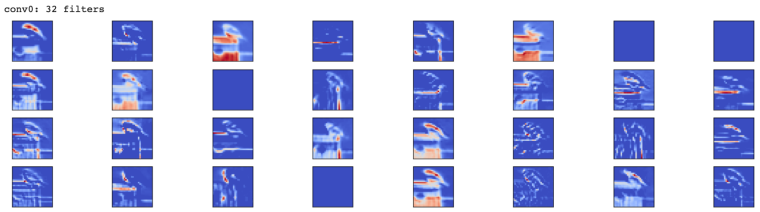
(d) Third conv output



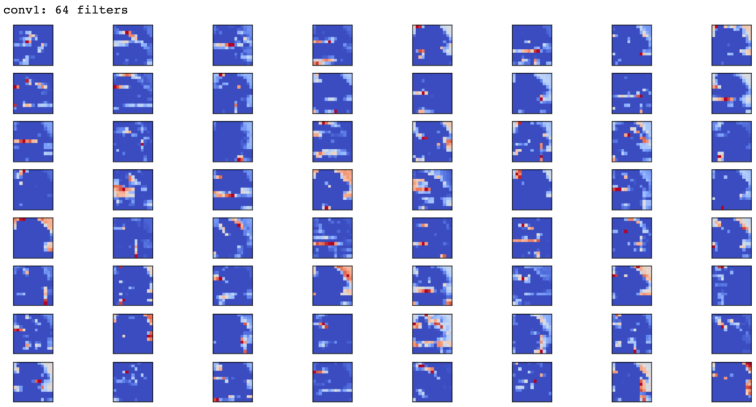
(e) Fourth conv output



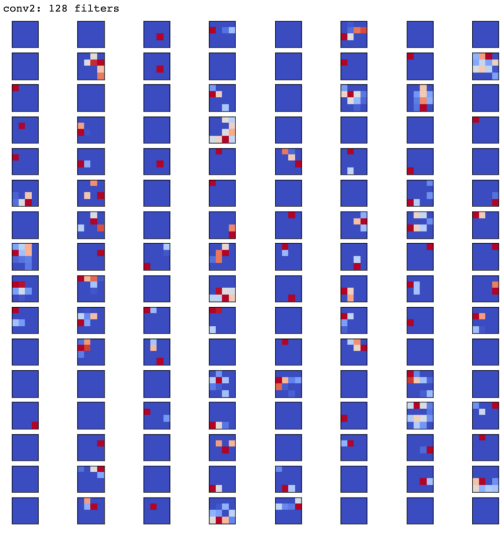
(a) Photo



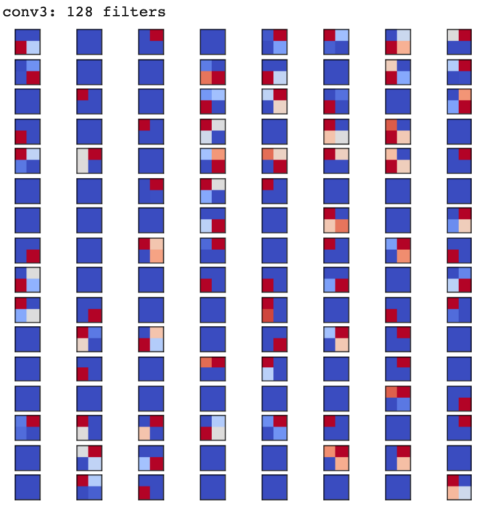
(b) First conv output



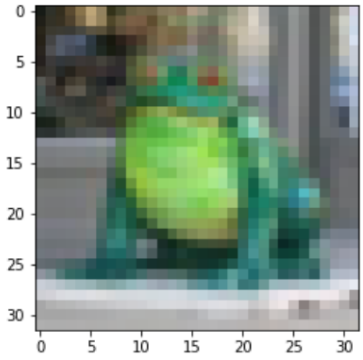
(c) Second conv output



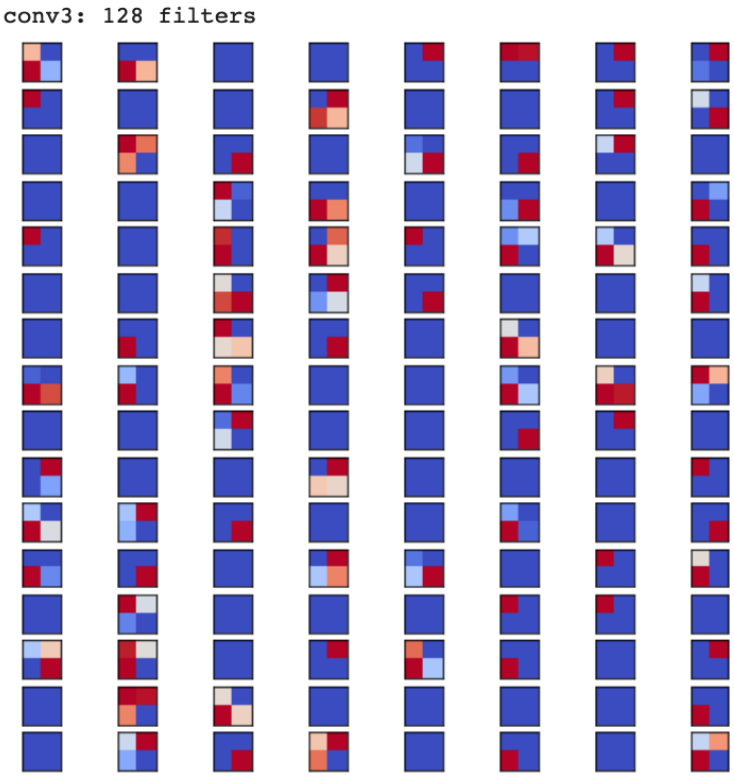
(d) Third conv output



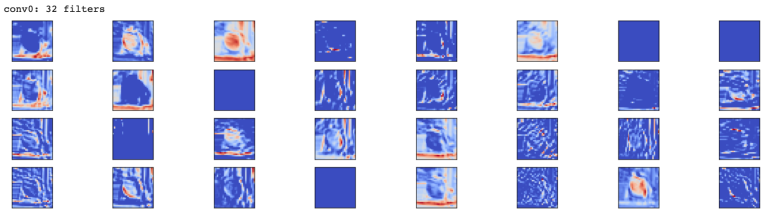
(e) Fourth conv output



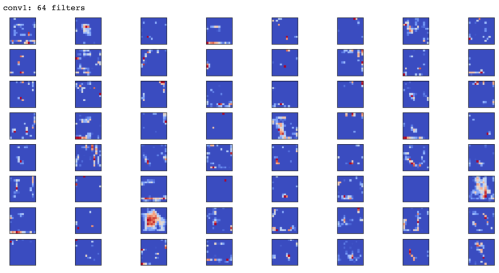
(a) Photo



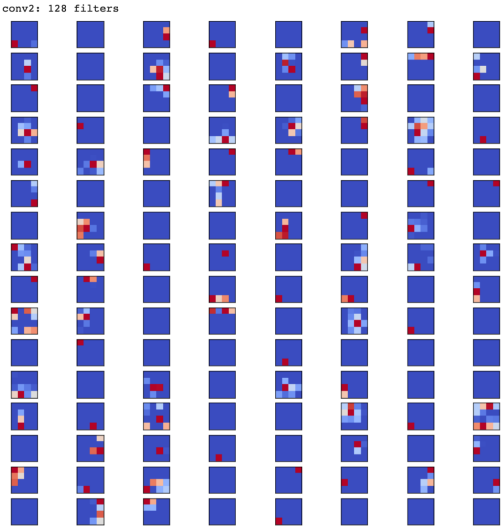
(b) First conv output



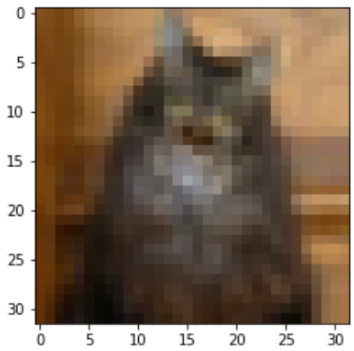
(c) Second conv output



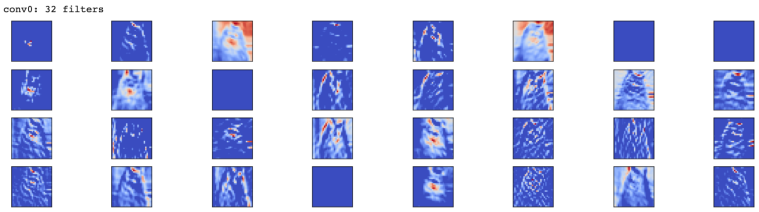
(d) Third conv output



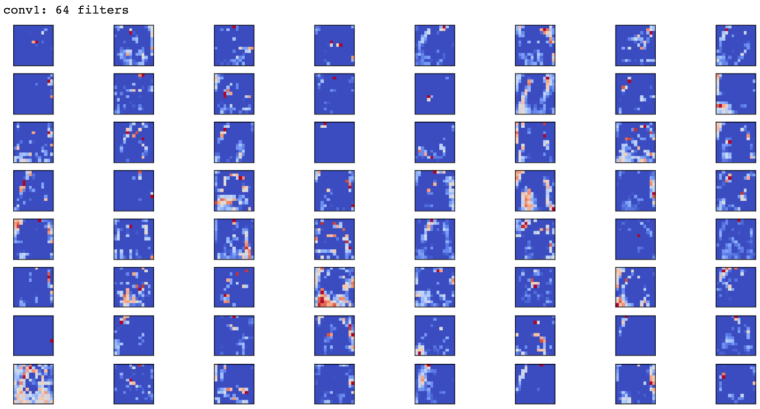
(e) Fourth conv output



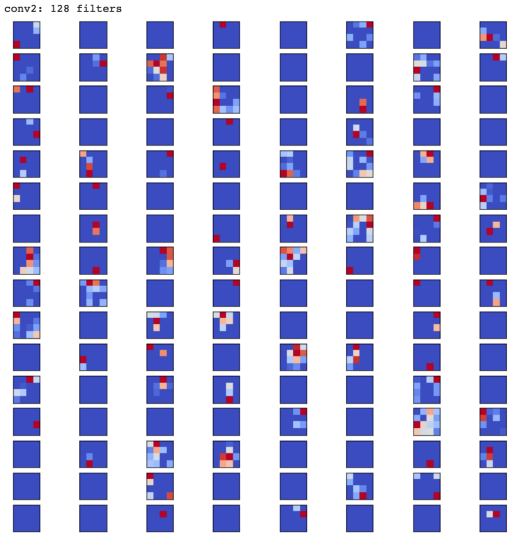
(a) Photo



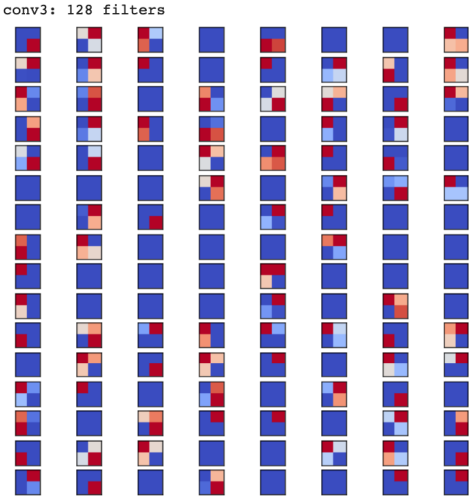
(b) First conv output



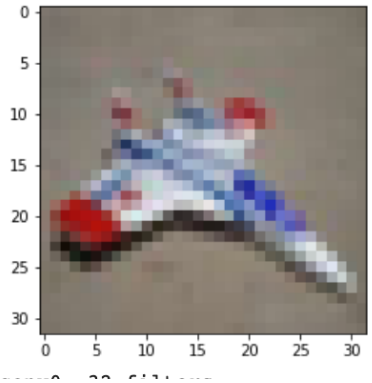
(c) Second conv output



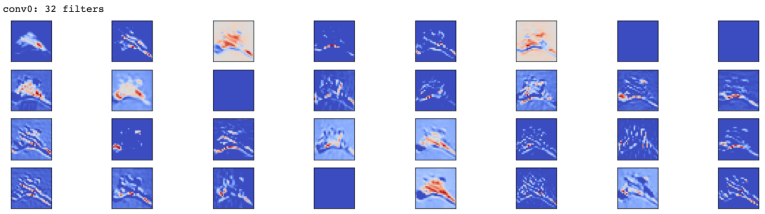
(d) Third conv output



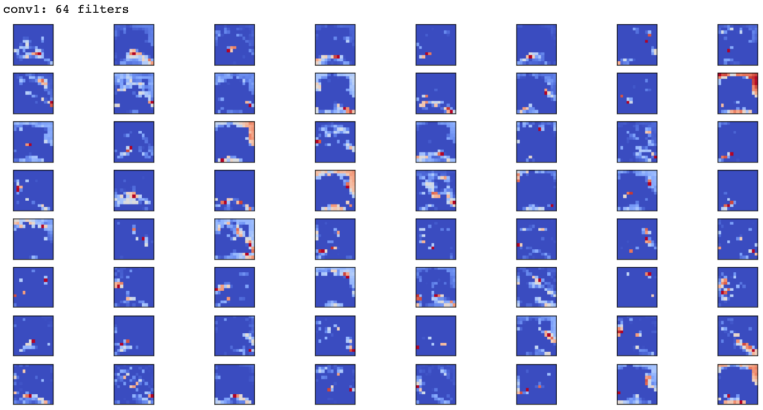
(e) Fourth conv output



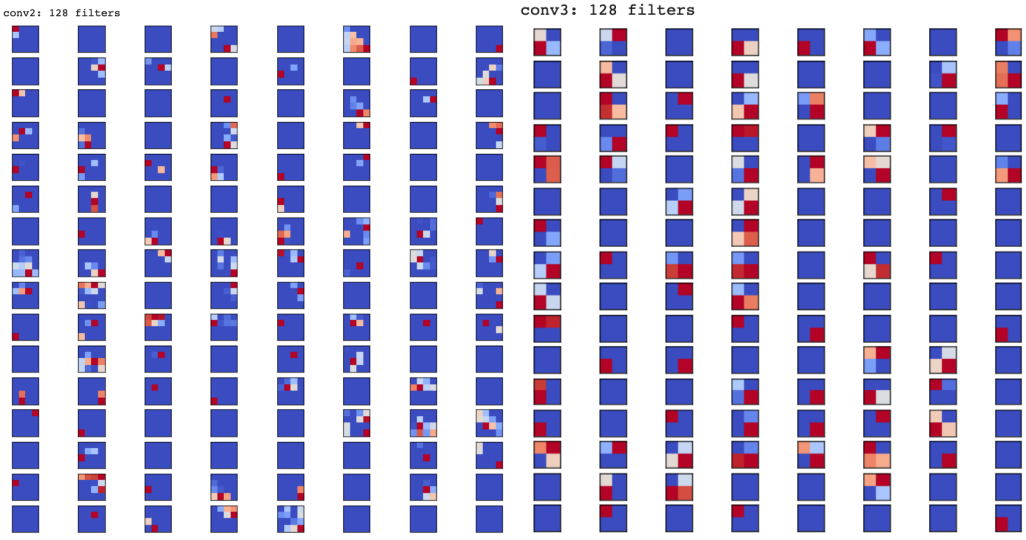
(a) Photo



(b) First conv output

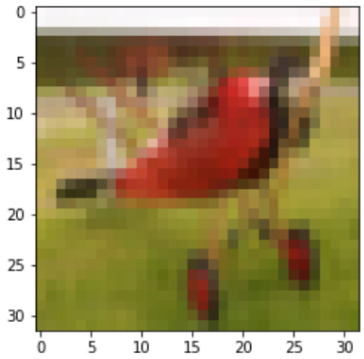


(c) Second conv output

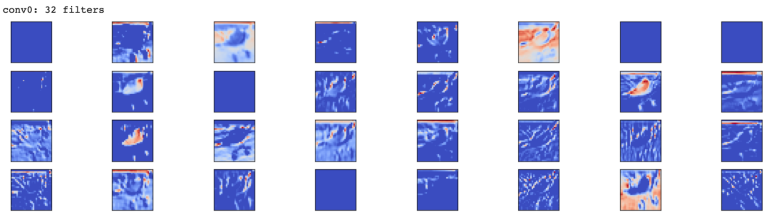


(d) Third conv output

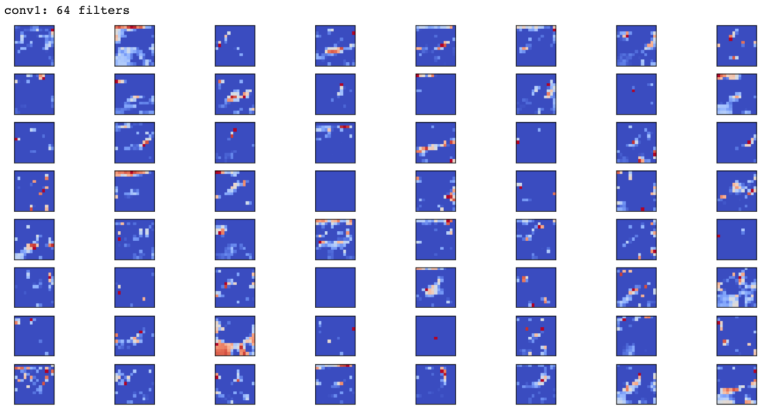
(e) Fourth conv output



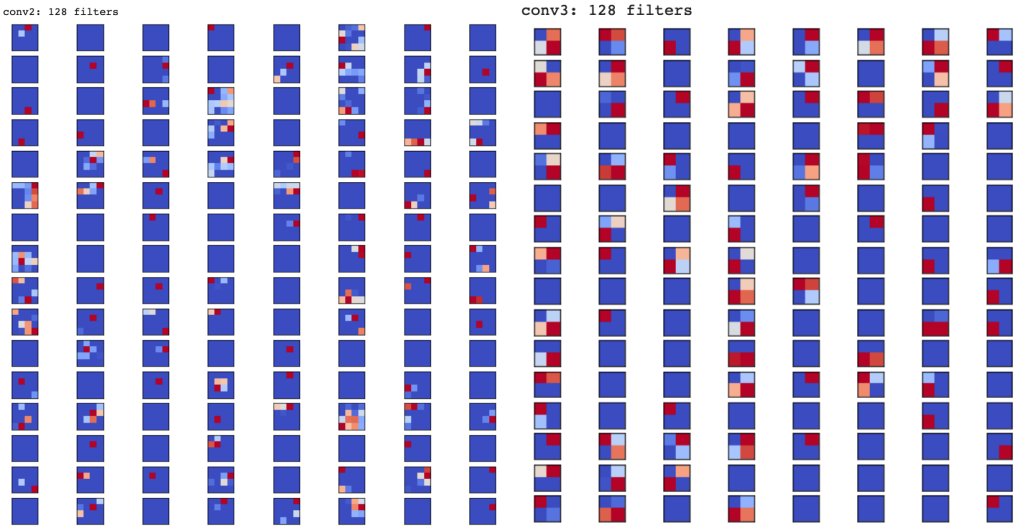
(a) Photo



(b) First conv output

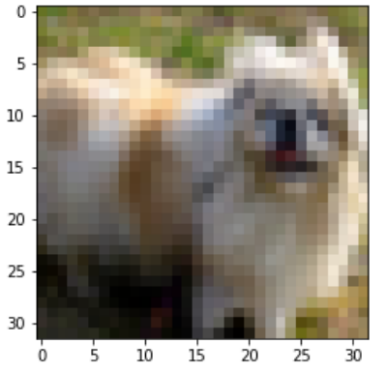


(c) Second conv output

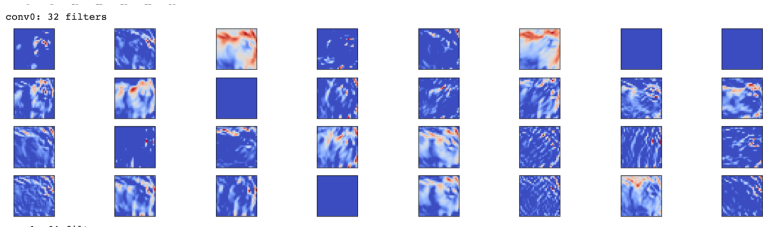


(d) Third conv output

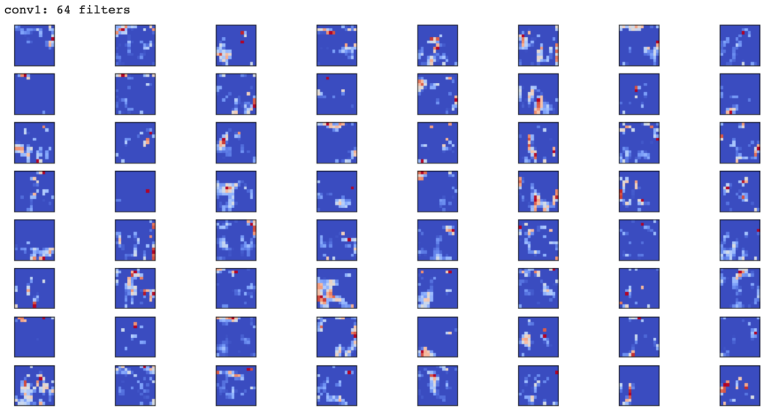
(e) Fourth conv output



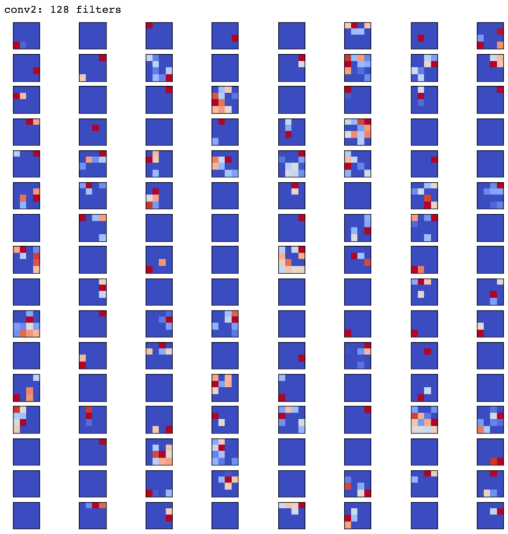
(a) Photo



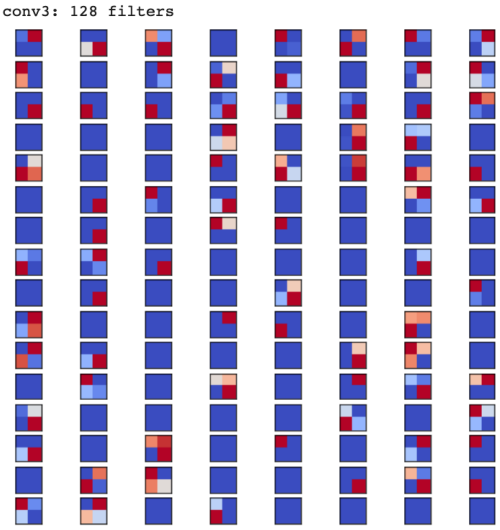
(b) First conv output



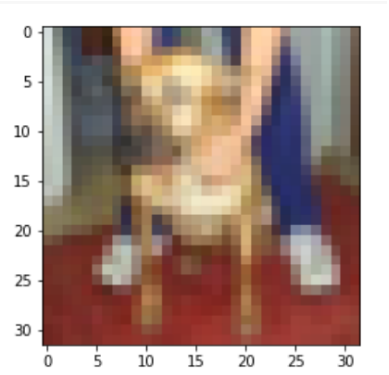
(c) Second conv output



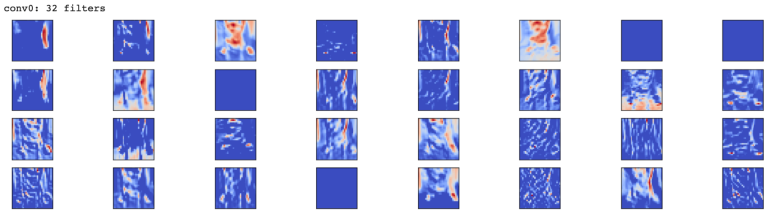
(d) Third conv output



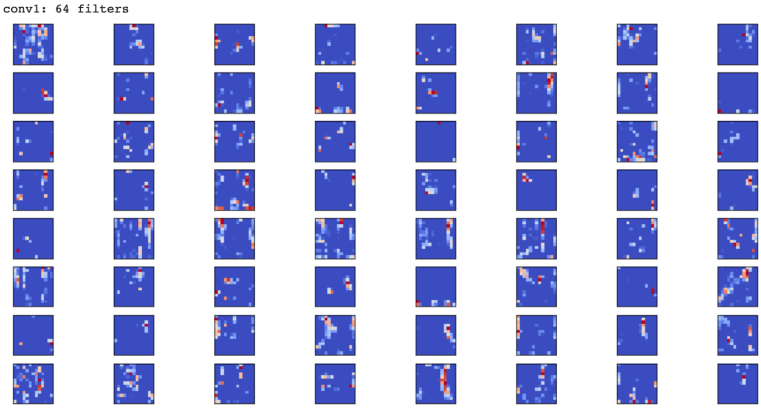
(e) Fourth conv output



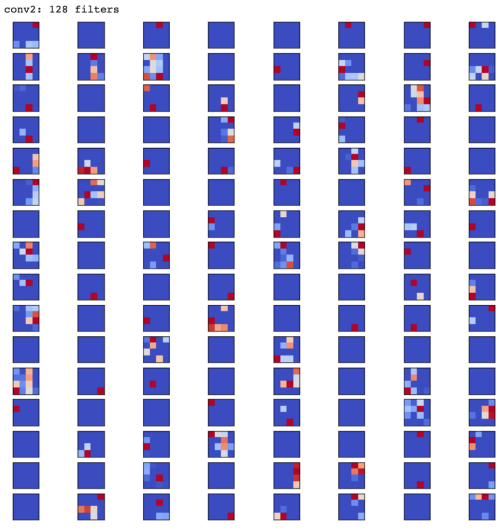
(a) Photo



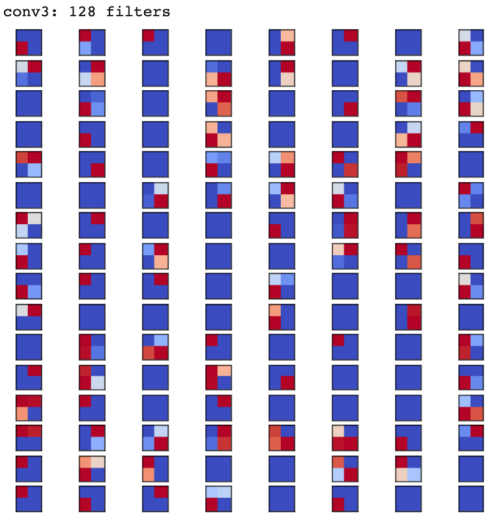
(b) First conv output



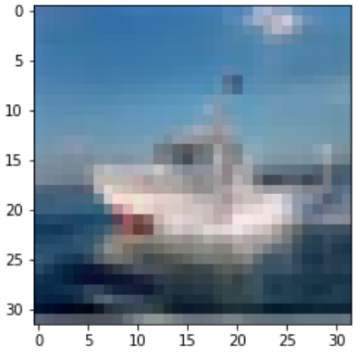
(c) Second conv output



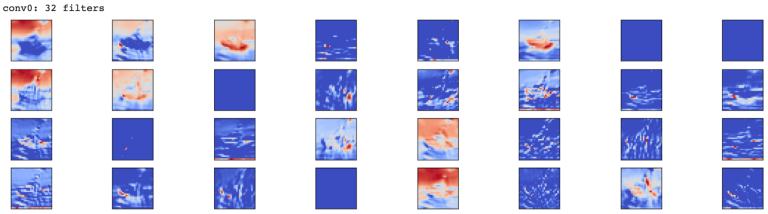
(d) Third conv output



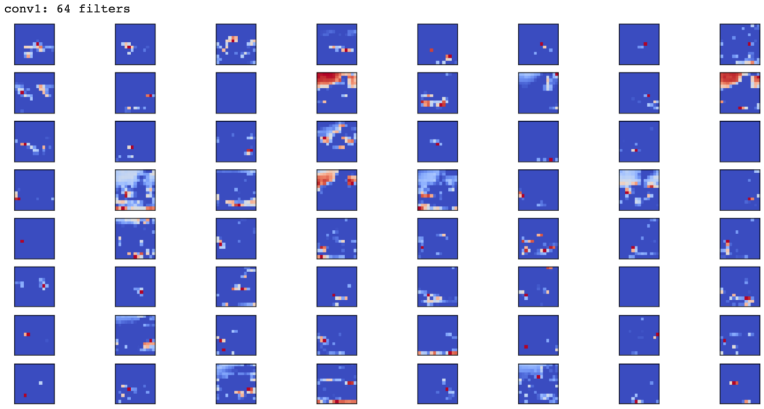
(e) Fourth conv output



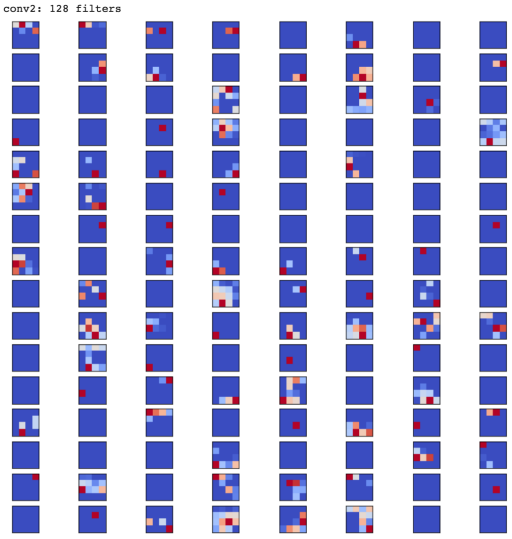
(a) Photo



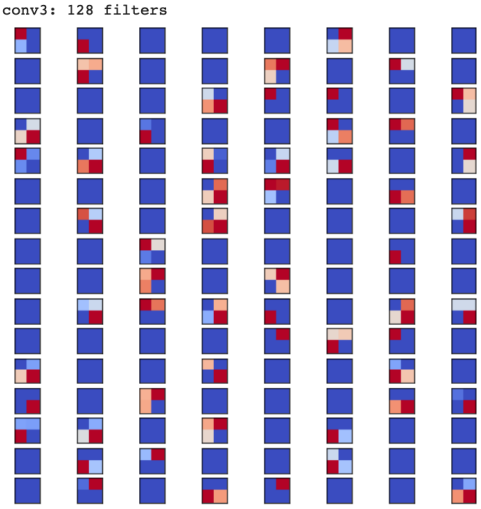
(b) First conv output



(c) Second conv output



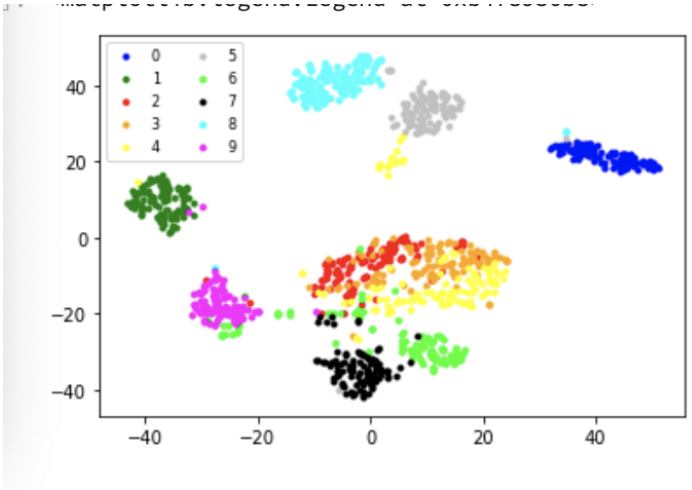
(d) Third conv output



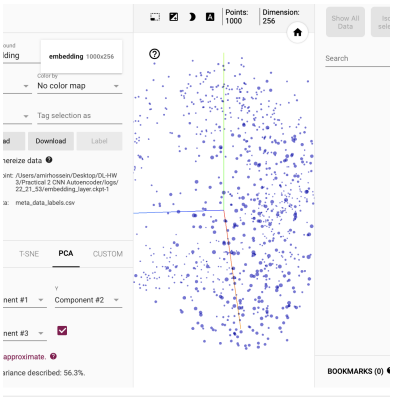
(e) Fourth conv output

Practical Excercise 2

The results are shown in bellow figure.



(a) TSNE



(b) Tensorboard PCA

(c) Tensorboard TSNE

(d) Tensorboard UMAP

```
    ax[1, i].imshow(np.squeeze(class_i_rotated[0]))
```

(a) Farsi OCR and Rotated farsi OCR

```
        _, v_loss = sess.run([model.opt, model.loss], feed_dict=feed_dict)
        total_loss += v_loss
    print('Epoch {} - Total loss: {}'.format(epoch + 1, total_loss))
```

```
Epoch 1 - Total loss: 1773.2327045798302
Epoch 2 - Total loss: 1129.7194606140256
Epoch 3 - Total loss: 965.8668376728892
Epoch 4 - Total loss: 888.5155786424875
Epoch 5 - Total loss: 839.9061506763101
```

Let's plot the reconstruction result for first batch of test set and compare them with their originals ones as well as their rotate

(b) Epoch loss

(c) Farsi OCR and Rotated farsi OCR and reconstructed OCR