Muhammad Yaseen (2577833)
Mohammad Molayem (2524422)
Dmitrii Badretdinov (2576757)

Question 1

Part a.

Eqns. 3-7 implemented in `two_layernet.py`.

Part b.

Code for calculating loss implemented in `loss()` function of `two_layernet.py`.

Part c.

The code was written using vectorized functions of `numpy`. Hence, no for-loops were used.

Question 2

Part a.

$$\frac{\partial J}{\partial z_k^{(3)}}(\theta, \{x_i, y_i\}_{i=1}^N) = ?$$

For the sake of simplicity in typing, we drop the coefficient ($\frac{1}{N}$) in J.

Consider $\omega_i = \frac{e^{z_i^{(3)}}}{\sum_j e^{z_j^{(3)}}}$.

Then:

$$log(\omega_i) = z_i^{(3)} - log(\sum_j e^{z_j^{(3)}})$$

We have

$$\begin{aligned}
\frac{\partial log(\omega_i)}{\partial z_k^{(3)}} &= \frac{\partial\Big(z_i^{(3)} - log(\sum_j e^{z_j^{(3)}})\Big)}{\partial z_k^{(3)}} \\
&= \delta_{ik} - \frac{e^{z_k^{(3)}}}{\sum_j e^{z_j^{(3)}}} \\
&= \delta_{ik} - \omega_k
\end{aligned}$$

where we have used the fact that

$$\frac{\partial \sum_j e^{z_j^{(3)}}}{\partial z_k^{(3)}} = e^{z_k^{(3)}}.$$

Putting all these back together

$$\begin{aligned}\frac{\partial J}{\partial z_k^{(3)}}(\theta, \{x_i, y_i\}_{i=1}^N) &= -\sum_i^N \frac{\partial log(\omega_i)}{\partial z_k^{(3)}} \\ &= -\sum_i^N (\delta_{ik} - \omega_k) \\ &= \sum_i^N (\omega_k - \delta_{ik}) \\ &= \sum_i^N (\psi(z_k^{(3)}) - \delta_{ik})\end{aligned}$$

Dropping the k index to make general (and bringing back the $\frac{1}{N}$ coefficient), we get

$$\boxed{\frac{\partial J}{\partial z^{(3)}}(\theta, \{x_i, y_i\}_{i=1}^N) = \frac{1}{N}(\psi(z^{(3)}) - \Delta),}$$

where

$$\Delta_{ij} = \begin{cases} 1 & \text{if } y_i = j \\ 0 & \text{if } otherwise \end{cases}$$

Part b.

Derivative of loss w.r.t. $W^{(2)}$

$$\frac{\partial J}{\partial W^{(2)}}(\theta, \{x_i, y_i\}_{i=1}^N) = ?$$

$$\begin{aligned}\frac{\partial J}{\partial W^{(2)}}(\theta, \{x_i, y_i\}_{i=1}^N) &= \frac{\partial J}{\partial z^{(3)}} \frac{\partial z^{(3)}}{\partial W^{(2)}} \\ &= \frac{1}{N}\Big(\psi(z^{(3)}) - \Delta\Big)\Big(\frac{\partial (W^{(2)} a^{(2)} + b^{(2)})}{\partial W^{(2)}}\Big)\end{aligned}$$

$$\boxed{\frac{\partial J}{\partial W^{(2)}}(\theta, \{x_i, y_i\}_{i=1}^N) = \frac{1}{N} a^{(2)T}\Big(\psi(z^{(3)}) - \Delta\Big)}$$

Derivative of regularized loss w.r.t. $W^{(2)}$:

$$\begin{aligned}\frac{\partial \tilde{J}}{\partial W^{(2)}}(\theta, \{x_i, y_i\}_{i=1}^N) &= \frac{\partial}{\partial W^{(2)}}\Big[J + \lambda\Big(\Big|\Big|W^{(1)}\Big|\Big|_2^2 + \Big|\Big|W^{(2)}\Big|\Big|_2^2\Big)\Big] \\ &= \frac{\partial J}{\partial W^{(2)}} + \frac{\partial\Big(\lambda\Big|\Big|W^{(1)}\Big|\Big|_2^2 + \lambda\Big|\Big|W^{(2)}\Big|\Big|_2^2\Big)}{\partial W^{(2)}} \\ &= \frac{1}{N} a^{(2)T}\Big(\psi(z^{(3)}) - \Delta\Big) + \lambda \sum_p \sum_q \frac{\partial W_{pq}^{(2)2}}{\partial W_{pq}^{(2)}} \\ &= \frac{1}{N} a^{(2)T}\Big(\psi(z^{(3)}) - \Delta\Big) + \lambda \sum_p \sum_q 2W_{pq}^{(2)}\end{aligned}$$

$$\boxed{\frac{\partial \tilde{J}}{\partial W^{(2)}}(\theta, \{x_i, y_i\}_{i=1}^N) = \frac{1}{N} a^{(2)T}\Big(\psi(z^{(3)}) - \Delta\Big) + 2\lambda W^{(2)}}$$

Part c.

Derivative of regularized loss w.r.t. $W^{(1)}$

$$\frac{\partial \tilde{J}}{\partial W^{(1)}} = \frac{\partial J}{\partial W^{(1)}} + 2\lambda W^{(1)}$$

$$\begin{aligned}\frac{\partial J}{\partial W^{(1)}} &= \frac{\partial J}{\partial z^{(3)}} \cdot \frac{\partial z^{(3)}}{\partial a^{(2)}} \cdot \frac{\partial a^{(2)}}{\partial W^{(1)}} \\ &= \frac{1}{N}\Big(\psi(z^{(3)}) - \Delta\Big) \cdot \frac{\partial z^{(3)}}{\partial a^{(2)}} \cdot \frac{\partial a^{(2)}}{\partial W^{(1)}}\end{aligned}$$

$$\frac{\partial z^{(3)}}{\partial a^{(2)}} = \frac{\partial W^{(2)} a^{(2)} + b^{(2)}}{\partial a^{(2)}} = W^{(2)}$$

$$\frac{\partial a^{(2)}}{\partial W^{(1)}} = \frac{\partial \phi(z^{(2)})}{\partial W^{(1)}} = a^{(1)} = x \qquad \text{if } z^{(2)} \geq 0\text{; else it is } 0$$

$$\boxed{\frac{\partial \tilde{J}}{\partial W^{(1)}} = \begin{cases} 2\lambda W^{(1)} + \frac{1}{N} a^{(1)T}\Big(\psi(z^{(3)}) - \Delta\Big) \cdot W^{(2)T} & \text{if} \quad z^{(2)} \geq 0 \\ 2\lambda W^{(1)} & \text{if} \quad z^{(2)} < 0 \end{cases}}$$

Derivative of regularized loss w.r.t. $b^{(1)}$:

$$\begin{aligned}\frac{\partial \tilde{J}}{\partial b^{(1)}} &= \frac{\partial J}{\partial b^{(1)}} + \lambda \frac{\partial\Big(\Big|\Big|W^{(1)}\Big|\Big|_2^2 + \Big|\Big|W^{(2)}\Big|\Big|_2^2\Big)}{\partial b^{(1)}} \\ &= \frac{\partial J}{\partial z^{(3)}} \cdot \frac{\partial z^{(3)}}{\partial b^{(1)}} \\ &= \frac{\partial J}{\partial z^{(3)}} \cdot \frac{\partial (W^{(2)} a^{(2)} + b^{(2)})}{\partial b^{(1)}} \\ &= W^{(2)} \frac{\partial J}{\partial z^{(3)}} \cdot \frac{\partial a^{(2)}}{\partial b^{(1)}} \\ &= W^{(2)} \frac{\partial J}{\partial z^{(3)}} \cdot \frac{\partial \phi^{(2)}}{\partial b^{(1)}} \\ &= W^{(2)} \frac{\partial J}{\partial z^{(3)}} \cdot \frac{\partial \phi(z^{(2)})}{\partial z^{(2)}} \cdot \frac{\partial z^{(2)}}{\partial b^{(1)}} \\ &= W^{(2)} \frac{\partial J}{\partial z^{(3)}} \cdot \frac{\partial \phi(z^{(2)})}{\partial z^{(2)}}\end{aligned}$$

$$\boxed{\frac{\partial \tilde{J}}{\partial b^{(1)}} = \begin{cases} \frac{W^{(2)}}{N}(\psi(z^{(3)}) - \Delta)^T & \text{if} \quad z^{(2)} \geq 0 \\ 0 & \text{if} \quad z^{(2)} < 0 \end{cases}}$$

Derivative of regularized loss w.r.t. $b^{(2)}$:

$$\begin{aligned}
\frac{\partial \tilde{J}}{\partial b^{(2)}} &= \frac{\partial J}{\partial b^{(2)}} + \lambda \frac{\partial \left(\left|\left| W^{(1)} \right|\right|_2^2 + \left|\left| W^{(2)} \right|\right|_2^2 \right)}{\partial b^{(2)}} \\
&= \frac{\partial J}{\partial b^{(2)}} \\
&= \frac{\partial J}{\partial z^{(3)}} \frac{\partial z^{(3)}}{\partial b^{(2)}} \\
&= \frac{\partial J}{\partial z^{(3)}} \frac{\partial (W^{(2)} a^{(2)} + b^{(2)})}{\partial b^{(2)}} \\
&= \frac{\partial J}{\partial z^{(3)}}
\end{aligned}$$

$$\boxed{\frac{\partial \tilde{J}}{\partial b^{(2)}} = \frac{1}{N}(\psi(z^{(3)}) - \Delta)}$$

Part d.

The backpropagation equations for network parameters $W^{(1)}$,$W^{(2)}$,$b^{(1)}$, and $b^{(2)}$ were implemented in the `loss()` function of `two_layernet.py` file. Using numerical gradients we also checked that the relative error for all the parameters is well within the allowed tolerance of 1e-8. A screen grab is show in Fig 1

Muhammad Yaseen (2577833)
Mohammad Molayem (2524422)
Dmitrii Badretdinov (2576757)

```
[yaseen:ex2]$ python ex2_FCnet.py
Your scores:
[[0.3644621  0.22911264 0.40642526]
 [0.47590629 0.17217039 0.35192332]
 [0.43035767 0.26164229 0.30800004]
 [0.41583127 0.2983228  0.28584593]
 [0.36328815 0.32279939 0.31391246]]

correct scores:
[[0.3644621  0.22911264 0.40642526]
 [0.47590629 0.17217039 0.35192332]
 [0.43035767 0.26164229 0.30800004]
 [0.41583127 0.2983228  0.28584593]
 [0.36328815 0.32279939 0.31391246]]

Difference between your scores and correct scores:
2.9173411492111612e-08
Difference between your loss and correct loss:
1.794120407794253e-13
W1 max relative error: 6.257855e-09
b1 max relative error: 1.250659e-09
W2 max relative error: 3.440708e-09
b2 max relative error: 3.865060e-11
Final training loss:  0.01714960793873205
```

Figure 1: Result of numerical gradient verification

Question 3

Part a.

SGD optimization algorithm with random mini-batch training was implemented in `two_layernet.py` file. During our execution it achieved a loss of 0.01715 on the toy dataset. See Fig. 2

Part b.

We first ran the training with the default hyperparameters settings and achieved a validation accuracy of 28.4%. We will refer to this model as **HLCV-Ex2-Baseline**. See Fig. 3 for the plots of loss vs. iteration, validation accuracy vs. epoch, and the visualization of learned features for **HLCV-Ex2-Baseline**.

After getting the baseline performance, we employed various approaches to adjust the hyperparameters to achieve higher classification accuracy. A discussion of each of these approaches and the results are given in the following paragraphs.

(1) Manual Changes: Experiments

- **Number of iterations**
  First intuition after seeing the baseline model was that the loss was decreasing but the training process was finished because `num_iters` has been set too low. We tried

Muhammad Yaseen (2577833)
Mohammad Molayem (2524422)
Dmitrii Badretdinov (2576757)

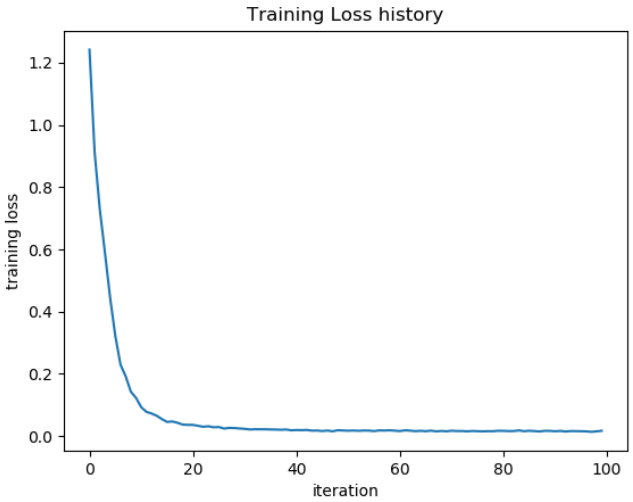

Figure 2: Loss curve on toy dataset

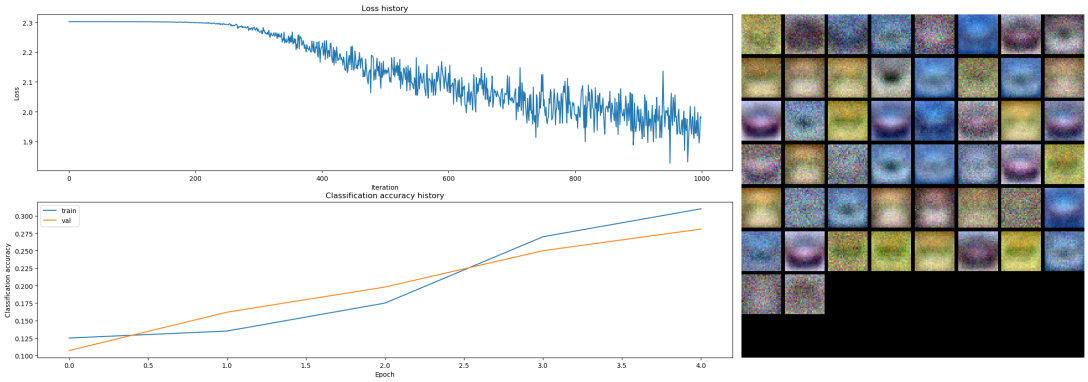

Figure 3: Loss vs. iteration (top-left), validation accuracy (bottom-left), and the visualization of learned features for HLCV-Ex2-Baseline.

Muhammad Yaseen (2577833)
Mohammad Molayem (2524422)
Dmitrii Badretdinov (2576757)

a range of iteration lengths to first make sure that baseline model has converged before moving on with more elaborate experimentation and tuning. The results of experimenting iteration lengths from 1000 to 10,000 have been presented in table 1. We see that the intuition was correct as model accuracy steadily increases on the validation set before flattening out after 8000.

Table 1: The effect of number of iterations on validation accuracy

Iterations	**Validation Accuracy**
1000	28.19%
2000	35.80%
3000	38.80%
4000	41.30%
5000	43.00%
6000	43.30%
7000	44.60%
8000	44.30%
9000	44.60%
10000	45.30%

- **Increasing Learning Rate**
  After having found the right number of iterations, the second obvious measure that one can take in order to enhance the validation accuracy is to try to reach to the convergence faster. To do this we increase the learning rate from 1e-3 to 1e-4. Results of this on the various iteration sizes are summarized in table 2.

Table 2: The Effect of increasing the learning rate on validation accuracy.

Iterations	**Validation Accuracy**
2000	49.90%
4000	52%
6000	51.60%
8000	51.60%
10000	52.60%

We see that we are able to achieve higher accuracy with about 4000 iterations. Although the case of 10000 iterations shows a slight improvement, we use 4000 iterations in subsequent experiments as a reasonable compromise between training speed and accuracy.

- **Number of Hidden Units**
  After adjusting the learning rate and training length to suitable values, we continued our experiment with model capacity, i.e. the number of units in the hidden layer. Results of this experiment are in table 3.

Table 3: The Effect of number of units in hidden layer on validation accuracy.

Number of Hidden Units	**Validation Accuracy**
10	42.60%
20	46.00%
30	48.60%
40	50.50%
50	**52.00**%
60	49.90%
70	50.60%
80	52.00%
90	51.50%
100	**53.90**%
110	52.90%
120	51.40%

We notice that model accuracy increases with the increase in capacity up to 50 hidden units, where it plateaus after. We experiment with higher values and find that 100 hidden units give the best performance, but we also note that this is only a marginal increase in accuracy upon doubling the model capacity. We continue experiments with these two nets. Hereafter, we refer to 50 units net as **HLCV-Ex2-50** and **HLCV-Ex2-100** respectively.

- **Batch Size**
  Now we try to identify the effect of different batch sizes on the performance. Results are summarized in table 4.

Table 4: The effect of batch size on the performance.

Number of Hidden Units	**Batch Size**	**Validation Accuracy**
50	100	48.80%
50	**200**	**52.00**%
50	300	49.50%
50	400	48.50%
50	500	50.80%
100	100	49.10%
100	**200**	**53.90**%
100	300	52.20%
100	400	52.00%
100	500	52.20%

We see that lower batch sizes result in more noisy gradients and hence loss, whereas as higher batch sizes are more smooth. There is no improvement in the accuracy for both **HLCV-Ex2-50** and **HLCV-Ex2-100**. The accuracy is actually lower in all other cases compared to 200 that was used in **HLCV-Ex2-Baseline**.

- **Regularization Multiplier λ**
  We now analyze the effect of regularization multiplier on the accuracy. For this we try values from 0 (no regularization) to 1.0. The results are mentioned in the table 5.

Muhammad Yaseen (2577833)
Mohammad Molayem (2524422)
Dmitrii Badretdinov (2576757)

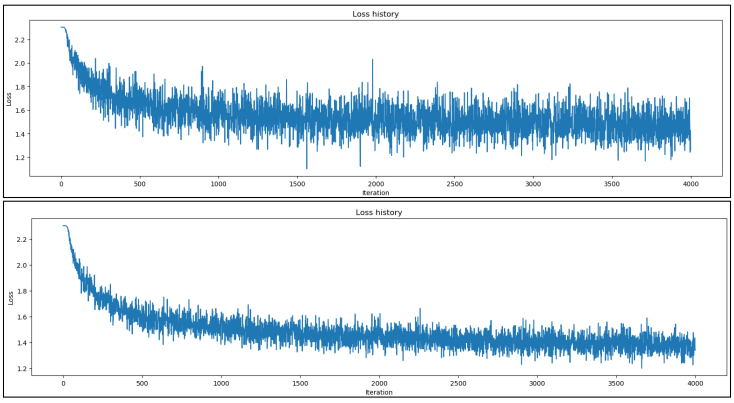

Figure 4: A smaller batch size i.e. 100 leads to wide variations in loss during training, whereas a higher batch size i.e. 300 results in smoother variations in loss.

Table 5: The effect of regularization on the validation accuracy for different number of hidden units.

Number of Hidden Units	**Regularization (λ)**	**Validation Accuracy**
50	0 (no regularization)	50.9%
50	**0.25**	**52%**
50	0.50	49.80%
50	0.75	50.40%
50	1.0	49.2%
100	**0** (no regularization)	**54.2%***
100	**0.25**	**53.90%**
100	0.50	52.70%
100	0.75	52.30%
100	1.0	49.70%

Again, we observe that there is no improvement in accuracy, except for the unregularized version (reg=0) of **HLCV-Ex2-100** network. But we still prefer the regularized version because of the understanding that it will have better generalization capability.

(2) Dropout

Lastly, we try the dropout regularization method, which randomly shuts down units in the hidden layer. The goal is to reduce over-fitting by not relying on any particular set of units being present. Results are summarized in table 6.

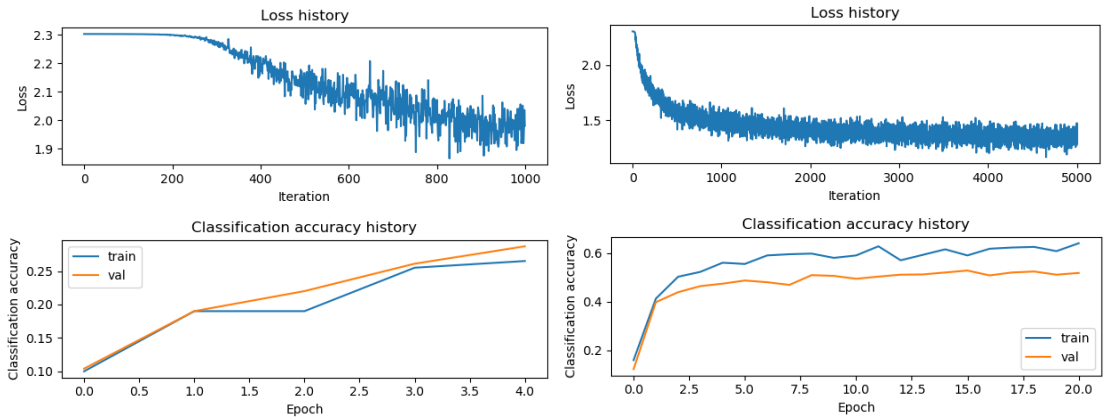

Figure 5: Loss vs iterations (top) and classification accuracy vs epoch (bottom) for cases with 28% (left) and 53% (right) validation accuracies. First we adjusted the hyperparameters by experiments (i.e. manual changes).

Table 6: The effect of Dropout on the validation accuracy for different number of hidden units.

Number of Hidden Units	**Keep Probability**	**Validation Accuracy**
50	0.25	34.00%
50	0.50	40.30%
50	0.75	46.50%
50	**1.0** (no dropout)	**52.00%**
100	0.25	37.60%
100	0.50	44.00%
100	0.75	46.20%
100	**1.0** (no dropout)	**53.90%**

As we saw in above section, this does not help with achieving any better accuracy. In fact, we see a severe degradation in performance due to dropout shutting down neurons and hence limiting model capacity. The training and validation curves now overlap, but model's capacity is limited.

(3) Dimension reduction: PCA

Another approach to improve the accuracy of the network was to perform Principal component analysis (PCA) and reduce the dimensionality. We used the PCA as implemented in *scikit-learn*. First, we plotted the cumulative summation of the explained variance versus the number of the principle components. From this curve, shown in the left panel of Fig. 6, one can see that if 95% of the variance is needed to be retained, it will be enough to have 250 components. Accordingly, we reduced the dimension. However, the experiment showed that PCA did not result in the classification accuracy enhancement (see table 7 and compare to the previous tables).

Muhammad Yaseen (2577833)
Mohammad Molayem (2524422)
Dmitrii Badretdinov (2576757)

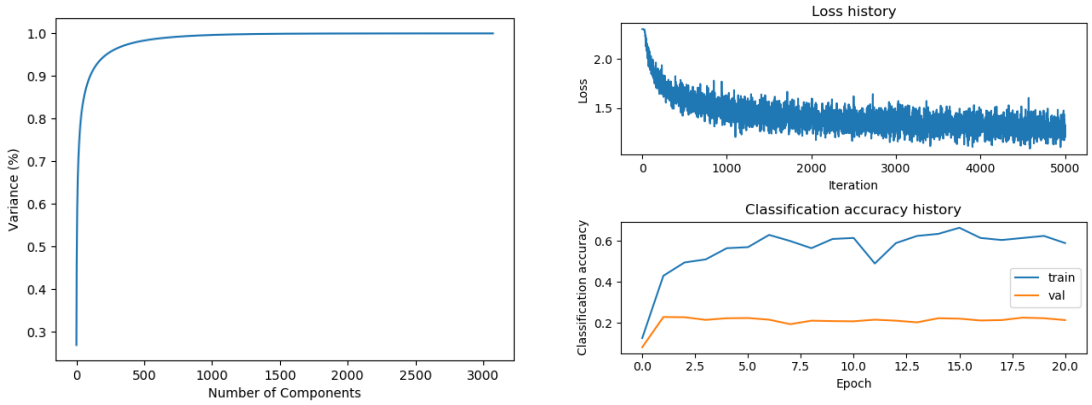

Figure 6: Left: The cumulative summation of the explained variance versus the number of the principle components. The curve shows that 95% of variance is retained by about 250 components. Right: the dimension reduction did not lead to a real enhancement in the classification accuracy.

	Input size	Neurons	Batch size	Iters.	Learn. rate	reg	Validation Acc. %
PCA	250	50	200	1000	10^{-4}	0.25	24.0
	250	50	400	5000	10^{-3}	0.3	21.6

Table 7: Validation accuracies after dimension reduction using PCA for two different sets of parameters. The second set in the table resulted in large values of accuracy ($\sim 50\%$) when no PCA was performed.

Conclusion:

After performing the experiments we choose the following parameters:

- Number of training iterations: 4000
- Batch Size: 200
- Learning Rate: 1e-3
- Hidden units: 100
- $\lambda = 0.25$
- No Dropout
- No PCA

We select HLCV-Ex2-100 as our best network. The test accuracy on this network is 53.8%. Visualization of the features learned by this network i.e HLCV-Ex2-100, and its close contender HLCV-Ex2-50 are shown in Fig. 7. A comparison with HLCV-Ex2-Baseline is interesting. We clearly see more variety in learned shapes i.e. edges, curves, circles etc. as well as more color variations compared to HLCV-Ex2-Baseline.

Muhammad Yaseen (2577833)
Mohammad Molayem (2524422)
Dmitrii Badretdinov (2576757)

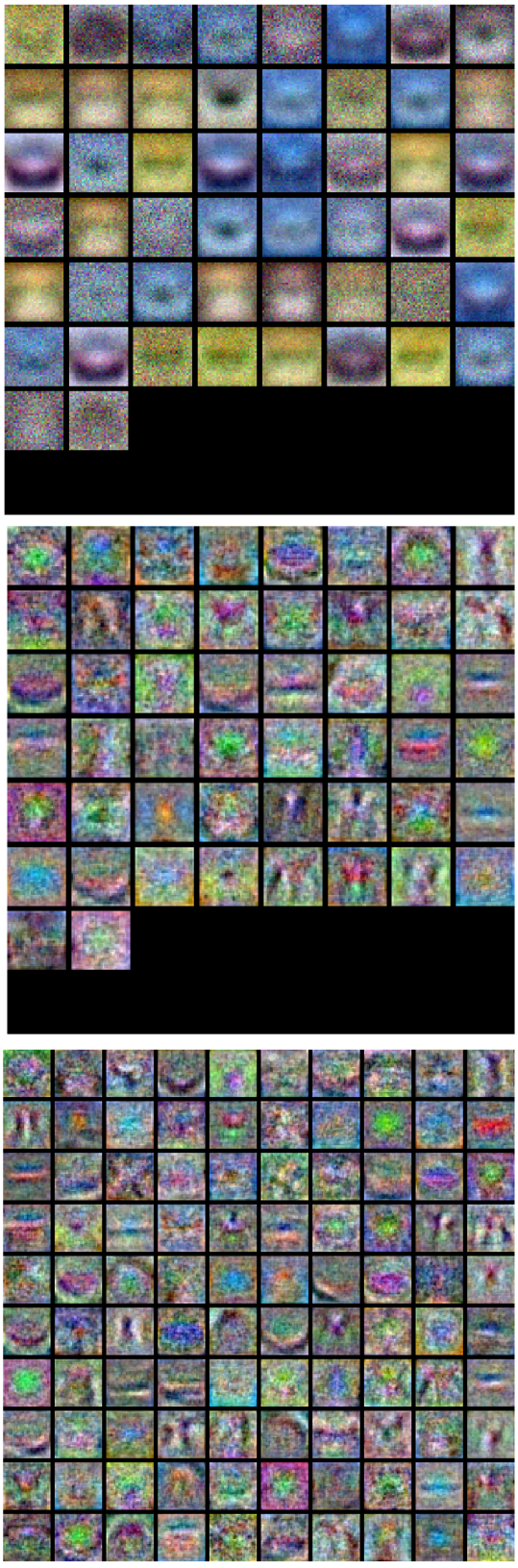

Figure 7: Features learned by HLCV-Ex2-Baseline (top), HLCV-Ex2-50 (middle), and HLCV-Ex2-100 (middle) visualized as an image.

Question 4

Part c.

A different number of the combination "Linear-ReLU" was tested.

The results are as follows:

Network type	Validation accuracy, %
2-layered network	52
3-layered network	48,6
4-layered network	7,8
5-layered network	7.8

2 layers-network has reached 52% accuracy on the validation set,
3 layers reached peaked at ninth epoch with 50.8% but went to 48.6% at 10th. Before the 9th epoch, a steady growth was shown.
4 layers started from 11% and went to the slightly changing 7.8% during later epochs.
5 layers has shown a performance which was very similar to the 4-layered version.

Based on the results, we can assume that the models with 4 and 5 linear layers have overfit the data to such a degree that a random choice would fair better than these network on the validation set.

The reason behind it is that the capacity of these two models was considerably higher when compared to the 3-layered model. Due to the insufficient amount of data, a system with such high capacity was not able to avoid overfitting.

The best network configuration (2 layers) has shown accuracy 51.3% on the 1000 test images.