Question1

Hyperparameters: Learning rate, batch size, and optimizer

Learning rates: [0.0001, 0.001, 0.01, 0.1]

Batch sizes: [512, 256, 128]

Optimizers: [SGD, adam, RMSProp]

Epoch: fixed 5

Regular CNN:

SGD showed the worst performance out of other optimizers. The best result from using SGD was the combination of learning rate 0.1 and batch size of 128.

```
Optimizer: SGD, Learning rate: 0.1, batch Size: 128
Epoch 1/5
469/469 [==============================] - 4s 7ms/step - loss: 1.2772 - accuracy: 0.5500
Epoch 2/5
469/469 [==============================] - 3s 7ms/step - loss: 0.1408 - accuracy: 0.9563
Epoch 3/5
469/469 [==============================] - 3s 7ms/step - loss: 0.0846 - accuracy: 0.9736
Epoch 4/5
469/469 [==============================] - 3s 7ms/step - loss: 0.0659 - accuracy: 0.9793
Epoch 5/5
469/469 [==============================] - 4s 8ms/step - loss: 0.0530 - accuracy: 0.9833
313/313 [==============================] - 1s 3ms/step - loss: 8.4617 - accuracy: 0.9791
Test loss: 8.461654663085938
Test accuracy: 0.9790999889373779
```

Adam showed the good performance. The best performance of the combination is that the learning rate is 0.001, and the batch size is 128.

```
Optimizer: adam, Learning rate: 0.001, batch Size: 128
Epoch 1/5
469/469 [==============================] - 4s 7ms/step - loss: 0.4051 - accuracy: 0.8676
Epoch 2/5
469/469 [==============================] - 3s 7ms/step - loss: 0.1012 - accuracy: 0.9698
Epoch 3/5
469/469 [==============================] - 3s 7ms/step - loss: 0.0692 - accuracy: 0.9791
Epoch 4/5
469/469 [==============================] - 3s 7ms/step - loss: 0.0542 - accuracy: 0.9836
Epoch 5/5
469/469 [==============================] - 3s 7ms/step - loss: 0.0463 - accuracy: 0.9857
313/313 [==============================] - 1s 3ms/step - loss: 6.7560 - accuracy: 0.9857
Test loss: 6.756014823913574
Test accuracy: 0.9857000112533569
```

RMSProp showed the best performance for the regular CNN model. The learning rate is 0.001 and the batch size is 128.

```
Optimizer: RMSProp, Learning rate: 0.001, batch Size: 128
Epoch 1/5
469/469 [==============================] - 5s 8ms/step - loss: 0.4086 - accuracy: 0.8687
Epoch 2/5
469/469 [==============================] - 4s 8ms/step - loss: 0.1018 - accuracy: 0.9690
Epoch 3/5
469/469 [==============================] - 4s 8ms/step - loss: 0.0686 - accuracy: 0.9791
Epoch 4/5
469/469 [==============================] - 4s 8ms/step - loss: 0.0520 - accuracy: 0.9840
Epoch 5/5
469/469 [==============================] - 4s 8ms/step - loss: 0.0414 - accuracy: 0.9869
313/313 [==============================] - 1s 3ms/step - loss: 4.9709 - accuracy: 0.9875
Test loss: 4.970912933349609
Test accuracy: 0.987500011920929
```

The performance of regular CNN is generally well. RMSProp optimizer with learning rate 0.001, and batch size 128 returned the best accuracy.

Inverted CNN

The best performance of SGD showed is 0.98 with SGD optimizer, 0.1 learning rate, and 128 batch size.

```
Optimizer: SGD, Learning rate: 0.1, batch Size: 128
Epoch 1/5
469/469 [==============================] - 5s 8ms/step - loss: 1.1635 - accuracy: 0.5825
Epoch 2/5
469/469 [==============================] - 4s 8ms/step - loss: 0.1400 - accuracy: 0.9567
Epoch 3/5
469/469 [==============================] - 4s 8ms/step - loss: 0.0879 - accuracy: 0.9727
Epoch 4/5
469/469 [==============================] - 4s 8ms/step - loss: 0.0708 - accuracy: 0.9784
Epoch 5/5
469/469 [==============================] - 4s 8ms/step - loss: 0.0563 - accuracy: 0.9822
313/313 [==============================] - 1s 3ms/step - loss: 8.5798 - accuracy: 0.9824
Test loss: 8.57982063293457
Test accuracy: 0.9824000000953674
```

Adam showed the best performance from the inverted CNN model. The learning rate is 0.001 and the batch size is 128.

```
Optimizer: adam, Learning rate: 0.001, batch Size: 128
Epoch 1/5
469/469 [==============================] - 5s 9ms/step - loss: 0.5765 - accuracy: 0.8128
Epoch 2/5
469/469 [==============================] - 4s 9ms/step - loss: 0.1420 - accuracy: 0.9574
Epoch 3/5
469/469 [==============================] - 4s 9ms/step - loss: 0.0975 - accuracy: 0.9709
Epoch 4/5
469/469 [==============================] - 5s 10ms/step - loss: 0.0748 - accuracy: 0.9773
Epoch 5/5
469/469 [==============================] - 4s 9ms/step - loss: 0.0655 - accuracy: 0.9808
313/313 [==============================] - 2s 5ms/step - loss: 7.8010 - accuracy: 0.9834
Test loss: 7.800996780395508
Test accuracy: 0.9833999872207642
```

RMSProp optimizer shows good performance in inverted CNN. The learning rate is 0.001 and the batch size is 128.

```
Optimizer: RMSProp, Learning rate: 0.001, batch Size: 128
Epoch 1/5
469/469 [==============================] - 6s 9ms/step - loss: 0.5995 - accuracy: 0.8107
Epoch 2/5
469/469 [==============================] - 4s 9ms/step - loss: 0.1438 - accuracy: 0.9583
Epoch 3/5
469/469 [==============================] - 4s 9ms/step - loss: 0.0884 - accuracy: 0.9736
Epoch 4/5
469/469 [==============================] - 4s 9ms/step - loss: 0.0697 - accuracy: 0.9791
Epoch 5/5
469/469 [==============================] - 4s 9ms/step - loss: 0.0547 - accuracy: 0.9832
313/313 [==============================] - 1s 3ms/step - loss: 6.2903 - accuracy: 0.9820
Test loss: 6.290252208709717
Test accuracy: 0.9819999933242798
```

The performance of inverted CNN is generally not well. Adam optimizer with learning rate 0.001, and batch size 128 returned the best accuracy.

Hour-glass shaped CNN

SGD showed the worst performance out of other optimizers. The best result from using SGD was the combination of learning rate 0.0001 and batch size of 128.

```
Optimizer: SGD, Learning rate: 0.0001, batch Size: 128
Epoch 1/5
469/469 [==============================] - 4s 7ms/step - loss: 2.2979 - accuracy: 0.1189
Epoch 2/5
469/469 [==============================] - 3s 7ms/step - loss: 2.0751 - accuracy: 0.2617
Epoch 3/5
469/469 [==============================] - 3s 7ms/step - loss: 0.5418 - accuracy: 0.8314
Epoch 4/5
469/469 [==============================] - 3s 7ms/step - loss: 0.2674 - accuracy: 0.9199
Epoch 5/5
469/469 [==============================] - 3s 7ms/step - loss: 0.2035 - accuracy: 0.9385
313/313 [==============================] - 1s 3ms/step - loss: 25.7139 - accuracy: 0.9482
Test loss: 25.71387481689453
Test accuracy: 0.948199987411499
```

Adam showed the good performance. The best performance of the combination is that the learning rate is 0.001, the batch size is 128.

```
Optimizer: adam, Learning rate: 0.001, batch Size: 128
Epoch 1/5
469/469 [==============================] - 4s 7ms/step - loss: 0.5364 - accuracy: 0.8263
Epoch 2/5
469/469 [==============================] - 3s 7ms/step - loss: 0.1222 - accuracy: 0.9640
Epoch 3/5
469/469 [==============================] - 4s 8ms/step - loss: 0.0848 - accuracy: 0.9745
Epoch 4/5
469/469 [==============================] - 3s 7ms/step - loss: 0.0671 - accuracy: 0.9796
Epoch 5/5
469/469 [==============================] - 3s 7ms/step - loss: 0.0576 - accuracy: 0.9829
313/313 [==============================] - 1s 3ms/step - loss: 7.3455 - accuracy: 0.9859
Test loss: 7.345521926879883
Test accuracy: 0.9858999848365784
```

RMSProp showed the best performance for the hour glass CNN model. The learning rate is 0.001 and the batch size is 128.

```
Optimizer: RMSProp, Learning rate: 0.1, batch Size: 128
Epoch 1/5
469/469 [==============================] - 6s 9ms/step - loss: 0.5267 - accuracy: 0.8318
Epoch 2/5
469/469 [==============================] - 4s 8ms/step - loss: 0.1371 - accuracy: 0.9582
Epoch 3/5
469/469 [==============================] - 4s 8ms/step - loss: 0.0855 - accuracy: 0.9739
Epoch 4/5
```

469/469 [==============================] - 4s 8ms/step - loss: 0.0654 - accuracy: 0.9804
Epoch 5/5
469/469 [==============================] - 4s 8ms/step - loss: 0.0522 - accuracy: 0.9840
313/313 [==============================] - 1s 3ms/step - loss: 5.9353 - accuracy: 0.9861
Test loss: 5.935269355773926
Test accuracy: 0.9861000180244446

The values of hyperparameters are:
Optimizer: SGD, Adam, RMSProp
Learning rate: 0.0001, 0.001, 0.01, 0.1
Batch size: 128, 256, 512

From the result, we can conclude that regular CNN performed better than other CNNs. The reason I believe the regular CNN performed better is that other CNN has some part of the layer uses less filter size as the training goes on. By decreasing the size of filter while the training is happening, it is failing to condense the features which is returning bad performance. Hyperparameters: The learning rate of 0.001 and batch size of 128 returns the highest accuracy. In addition, I have found that adam and RMSProp optimizer are averagely performs better than SGD.

Question2

1. What is the effect of learning rate on the training process? Which performed best?

   Learning rate: 0.0001
   Test loss: 1.3807892799377441
   Test accuracy: 0.5338000059127808

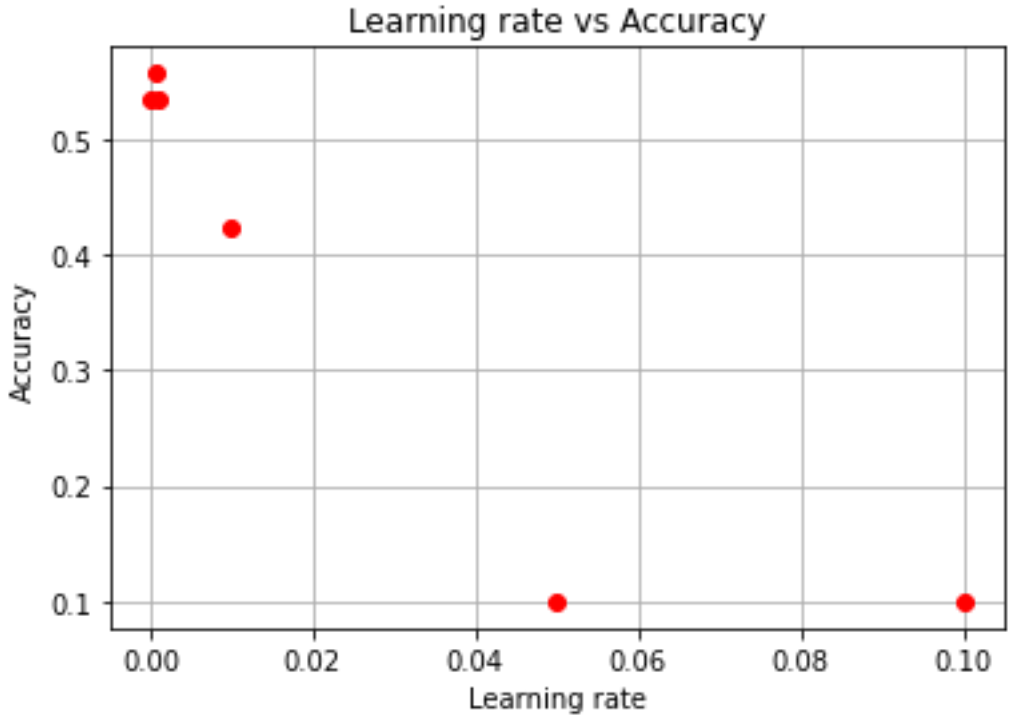
   Learning rate: 0.0005
   Test loss: 2.2831778526306152
   Test accuracy: 0.5565999746322632

   Learning rate: 0.001
   Test loss: 2.6412055492401123
   Test accuracy: 0.534600019454956

   Learning rate: 0.01
   Test loss: 1.6260498762130737
   Test accuracy: 0.4242999851703644

   Learning rate: 0.05
   Test loss: 2.3036272525787354
   Test accuracy: 0.10000000149011612

   Learning rate: 0.1
   Test loss: 2.304857015609741
   Test accuracy: 0.10000000149011612



The best performed learning rate is 0.0005.
From the result we can find that the higher learning rate decrease accuracy for this model. This means that the higher learning rate is causing the model to fail to detect local minima in the middle of training process.

2. What is the effect of batch size on the training process? Which performed best?

   Batch Size: 32
   Test loss: 3.1822121143341064
   Test accuracy: 0.5819000005722046

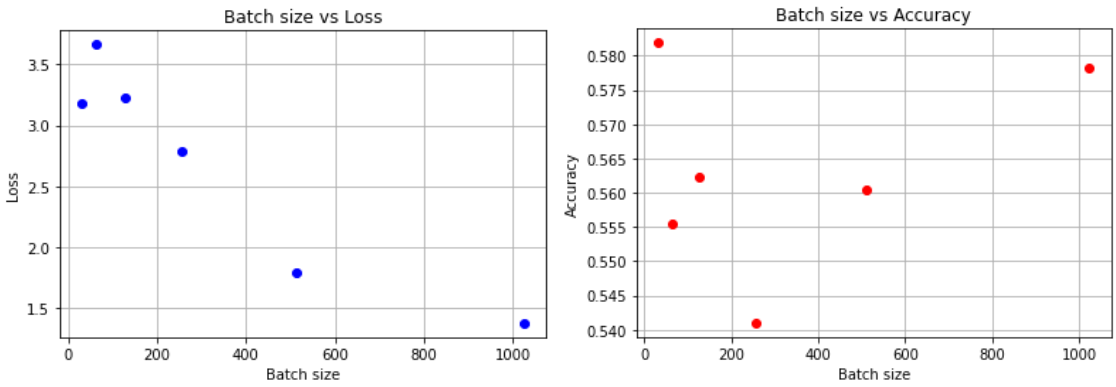
   Batch Size: 64
   Test loss: 3.661105155944824
   Test accuracy: 0.555400013923645

   Batch Size: 128
   Test loss: 3.225025177001953
   Test accuracy: 0.5622000098228455

   Batch Size: 256
   Test loss: 2.786754608154297
   Test accuracy: 0.5410000085830688

   Batch Size: 512
   Test loss: 1.7933979034423828
   Test accuracy: 0.5605000257492065

   Batch Size: 1024
   Test loss: 1.3790733814239502
   Test accuracy: 0.5781999826431274



As the batch size increase, the loss is decreasing, and the accuracy of the model is increasing. Even though the low batch size has good accuracy however, it has high loss score, therefore, high batch size with lower loss score represents the better performance. From this result, we can say that the larger batch size tends to be better.

3. Try different hyperparameters to obtain the best accuracy on the test set. What is your best performance and what were the hyperparameters?

Epoch: 10
Test loss: 1.341706395149231
Test accuracy: 0.5257999897003174

Epoch: 20
Test loss: 1.3193355798721313
Test accuracy: 0.5698000192642212

Epoch: 30
Test loss: 1.9635429382324219
Test accuracy: 0.5583000183105469

Epoch: 40
Test loss: 2.3796188831329346
Test accuracy: 0.5475000143051147

Epoch: 50
Test loss: 2.734956741333008
Test accuracy: 0.5764999985694885

The other hyperparameter I tried is the epoch. More iteration seems like giving better accuracy but also increases the total loss.

The best combination of hyperparameters is batch size of 1024, learning rate of 0.0005 and epoch of 25.
Test loss: 1.3790733814239502
Test accuracy: 0.5781999826431274

4. Implement an equivalent feed forward network for the same task with each hidden layer containing the same number of neurons as the number of filters in each convolution layer. Use the 'Adam' optimizer to train your network on the CIFAR-10 dataset for a fixed set of 25 epochs. Compare its performance with your LeNet implementation based on the following questions:

a. What is its performance?

Epoch 1/25
49/49 [==============================] - 11s 213ms/step - loss: 58.8145 - accuracy: 0.0984
Epoch 2/25
49/49 [==============================] - 11s 215ms/step - loss: 2.3026 - accuracy: 0.1000

. . .

Epoch 23/25
49/49 [==============================] - 10s 202ms/step - loss: 2.3025 - accuracy: 0.0976
Epoch 24/25
49/49 [==============================] - 10s 202ms/step - loss: 2.3025 - accuracy: 0.0998

Epoch 25/25
49/49 [==============================] - 10s 202ms/step - loss: 2.3025 - accuracy: 0.0975

Test loss: 2.303964614868164
Test accuracy: 0.10000000149011612

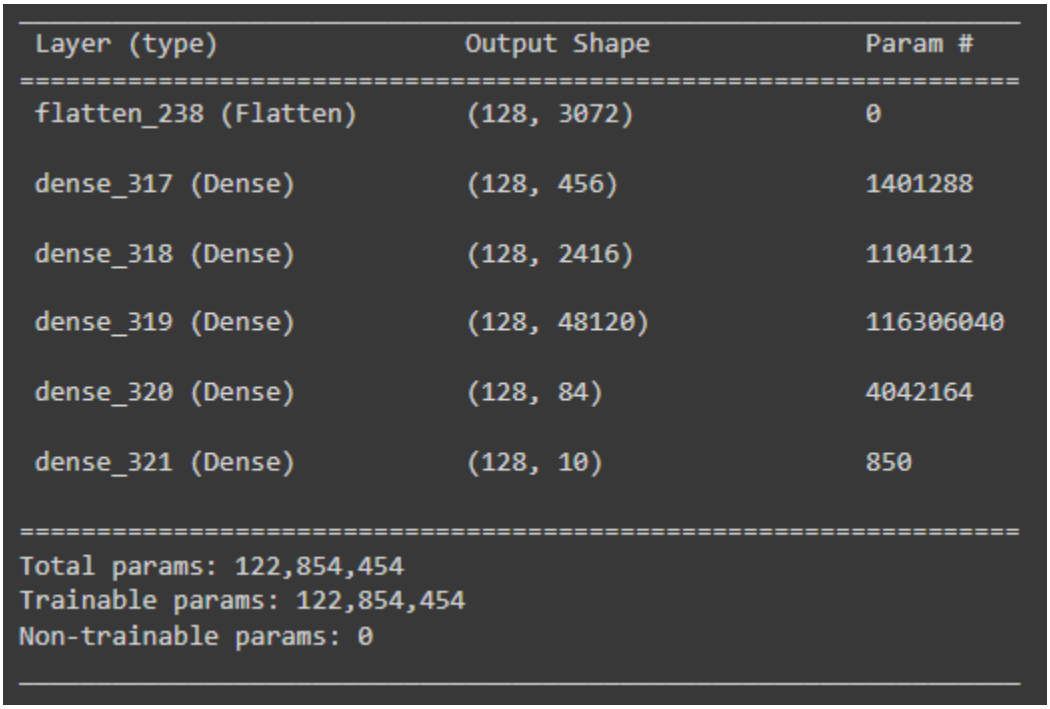
The performance of the feed forward network is that it stop improves immediately right after the first epoch and defined in loss 2.3025 and accuracy 0.09. The performance of the feed forward network compared with the CNN shows that the CNN has better detection of features to increase the performance.

b. How many parameters are there in this network compared to the LeNet implementation? Are they worth it?

LeNet has 697,046 total parameters.

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_1950 (Conv2D)        (128, 32, 32, 6)          456

 max_pooling2d_820 (MaxPooli  (128, 16, 16, 6)         0
 ng2D)

 conv2d_1951 (Conv2D)        (128, 16, 16, 16)         2416

 max_pooling2d_821 (MaxPooli  (128, 8, 8, 16)          0
 ng2D)

 conv2d_1952 (Conv2D)        (128, 8, 8, 120)          48120

 flatten_230 (Flatten)       (128, 7680)               0

 dense_280 (Dense)           (128, 84)                 645204

 dense_281 (Dense)           (128, 10)                 850

=================================================================
Total params: 697,046
Trainable params: 697,046
Non-trainable params: 0
_________________________________________________________________
```

Feed Forward network has 122,854,454 total parameters.



Layer (type)	Output Shape	Param #
flatten_238 (Flatten)	(128, 3072)	0
dense_317 (Dense)	(128, 456)	1401288
dense_318 (Dense)	(128, 2416)	1104112
dense_319 (Dense)	(128, 48120)	116306040
dense_320 (Dense)	(128, 84)	4042164
dense_321 (Dense)	(128, 10)	850

Total params: 122,854,454
Trainable params: 122,854,454
Non-trainable params: 0

Feed forward network has more parameters than LeNet but show worse performance than LeNet. We can conclude that it was not worth to have more parameters.

Question3

1. What are the dimensions of the input and the kernel (or filter)? How many parameters are there in the kernel f? [2 points]

```
Dimensions of the input: 6 X 6 X 1
(6, 6)
[[7 5 0 0 3 2]
 [6 4 5 1 4 8]
 [9 0 2 2 5 4]
 [6 3 4 7 9 8]
 [5 7 5 6 9 0]
 [7 9 0 8 2 3]]

Dimensions of the filter: 3 X 3 X 1
(3, 3)
[[ 1  0 -1]
 [ 2  0 -2]
 [ 1  0 -1]]
```

The input has volume of 6 * 6 *1 and the filter have volume of 3 * 3 * 1.
Since the depth of input is 1, filter depth is also 1.
Therefore, the dimensions of the input and the kernel are 1.

The parameters for the kernel f can be achieved as ((3 * 3 * 1) + 1) * 1 = 10.

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d_1 (Conv2D)            (1, 4, 4, 1)              10

=================================================================
Total params: 10
Trainable params: 10
Non-trainable params: 0
_________________________________________________________________
```

2. What is the output activation map when you apply the convolutional operation using the filter f on the input X without padding? [4 points]

The output activation map can be achieved by calculating the sum of multiplication of two matrices (part of 3*3 input and filter) for each corresponding element of 4*4 output matrix. The code to achieve below output is available in the submitted python file.

```
[[ 16.   9.  -4. -18.]
 [ 17.  -5. -10. -12.]
 [ 11.  -9. -17.   2.]
 [  9.  -1. -15.  16.]]
```

3. What is the output when you apply a max-pooling operation on the output from the previous question? [4 points]

   The output of applying a max-pooling operation is:

```
[[17. -4.]
 [11. 16.]]
```

   This can be achieved with 2 stride and 2 filter size. It will separate the matrix into 4 parts without any overlapping areas. Each area will determine the max value to be represent the whole area. By decreasing the dimension, we can cut the calculation time.
   The code to achieve below output is available in the submitted python file.