A report on assignment 1

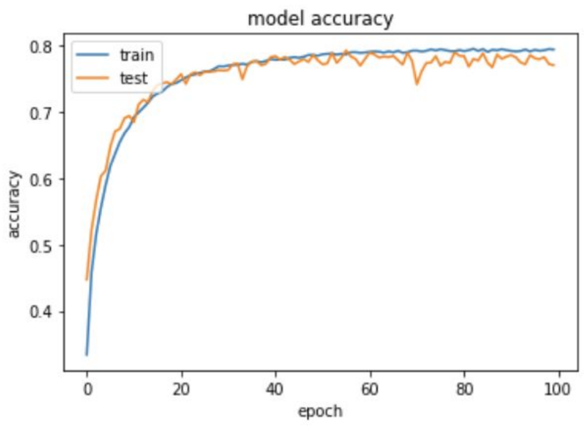
Problem 1:

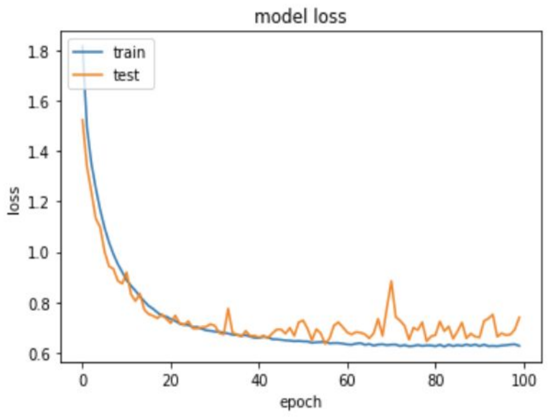
In this problem, a dataset of 60,000 records in which 10,000 records is test set is introduced. The dataset can be directly imported from keras, and needs no data preprocessing.

Upon a Sequential object from keras, the program adds different layers to build a MLP neural network. Based on the model shared(https://github.com/fchollet/keras/blob/master/examples/cifar10_cnn.py), the team finds that changes on the following parameters may influence the performance of the model:

1. Number of epochs.

Through 100 epochs the program plotted the curve of accuracy and loss. The team finds that no significant increase of performance is happening after 30 epochs (learning rate = 0.0001), and no there is an obvious overfitting issue rising before the 40th epoch. Thus though a little bit under-fitting, the team decided to put 30 epochs in the model.





2. Batch size

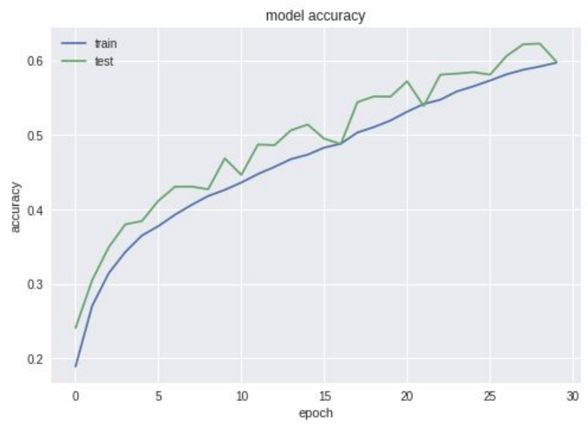
A batch size indicates the amount of records to be involved in back propagation. Thus, too large a batch size decrease the efficiency of back propagation, while too small a batch size may slow down the process as a whole.

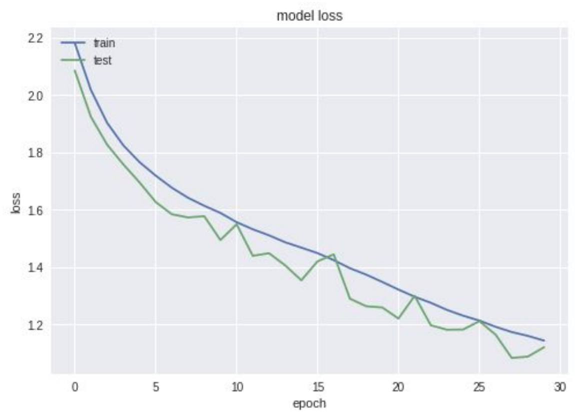
Under this very circumstance the team finds a trade off between the network performance and speed, using the following experiment:

When batch size is 32:

Test loss: 0.6967130443572999

Test accuracy: 0.7642





When batch size increased as 128:

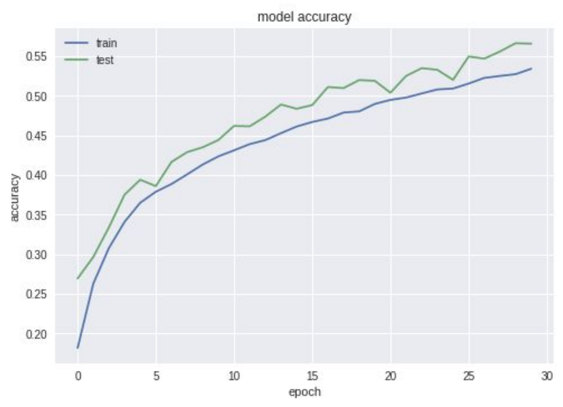
Test loss: 0.799194857120514, Test accuracy: 0.7223

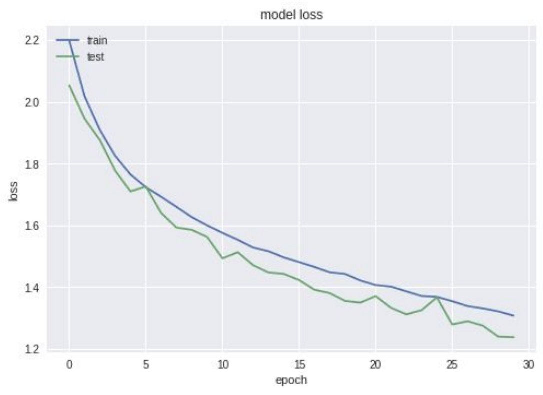
……

And when batch size increased to 256:

Test loss: 0.9514189182281494 Test accuracy: 0.6693

As for a batch size of 1024:

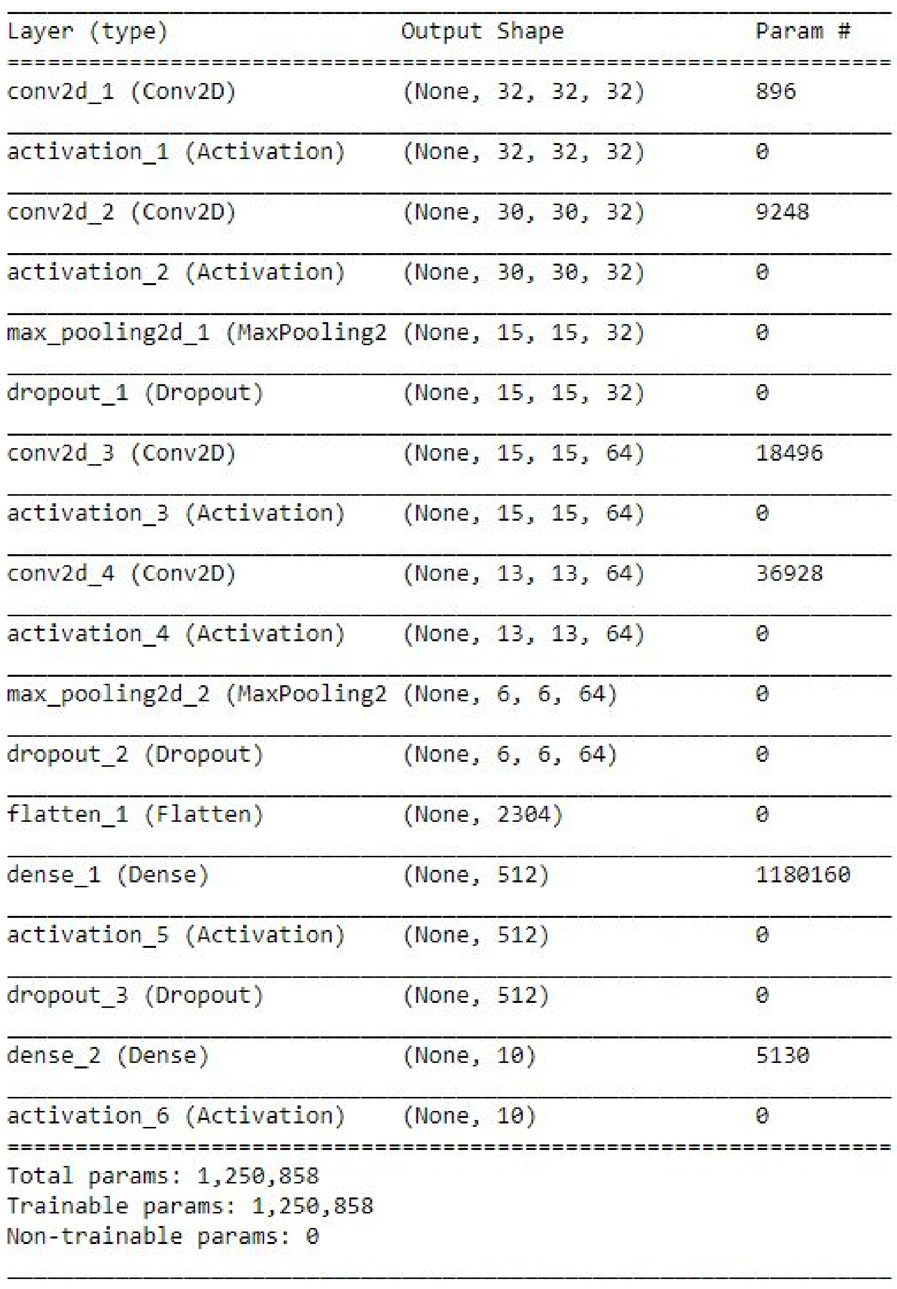




This leaves the model much fewer chances to improve itself, which leads an underfitting.

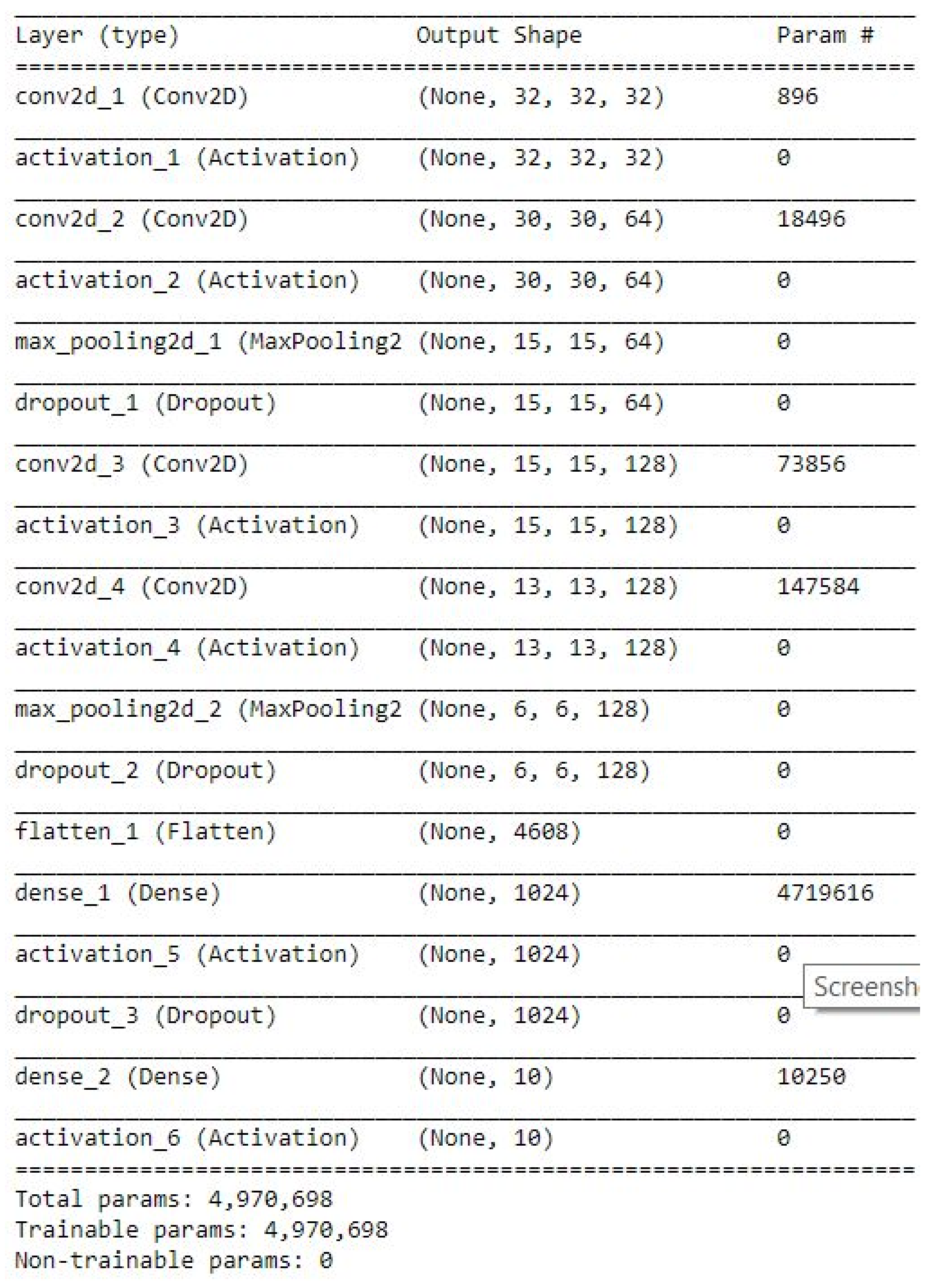
3. Network configuration:

The amount of layers and number of nodes together defines the complexity of a neural network. There is another trade off issue here between overfitting and underfitting when configuring the model.



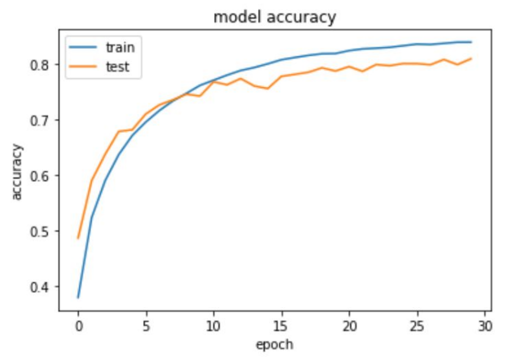
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 32, 32, 32)        896
_________________________________________________________________
activation_1 (Activation)    (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 30, 30, 32)        9248
_________________________________________________________________
activation_2 (Activation)    (None, 30, 30, 32)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 15, 15, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 15, 15, 64)        18496
_________________________________________________________________
activation_3 (Activation)    (None, 15, 15, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 13, 13, 64)        36928
_________________________________________________________________
activation_4 (Activation)    (None, 13, 13, 64)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
dropout_2 (Dropout)          (None, 6, 6, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 2304)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               1180160
_________________________________________________________________
activation_5 (Activation)    (None, 512)               0
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                5130
_________________________________________________________________
activation_6 (Activation)    (None, 10)                0
=================================================================
Total params: 1,250,858
Trainable params: 1,250,858
Non-trainable params: 0
_________________________________________________________________
```

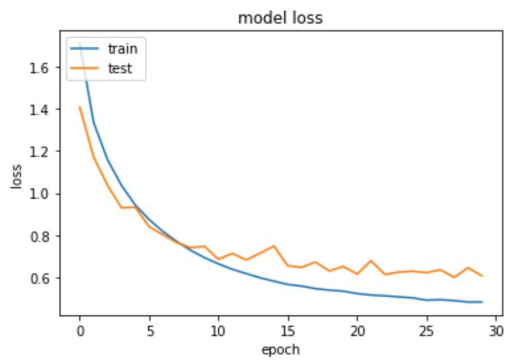
a. Number of neurons in a layer:
   Keeping other parameters the same as above, and changing following layers' neurons number: "conv2d_2" from 32 to 64, "conv2d_3": 64 to 128, "conv2d_4": 64 to 128, "dense_1": 512 to 1024



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 32, 32, 32)        896
_________________________________________________________________
activation_1 (Activation)    (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 30, 30, 64)        18496
_________________________________________________________________
activation_2 (Activation)    (None, 30, 30, 64)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 15, 15, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 15, 15, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 15, 15, 128)       73856
_________________________________________________________________
activation_3 (Activation)    (None, 15, 15, 128)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 13, 13, 128)       147584
_________________________________________________________________
activation_4 (Activation)    (None, 13, 13, 128)       0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 6, 6, 128)         0
_________________________________________________________________
dropout_2 (Dropout)          (None, 6, 6, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 4608)              0
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              4719616
_________________________________________________________________
activation_5 (Activation)    (None, 1024)              0
_________________________________________________________________
dropout_3 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                10250
_________________________________________________________________
activation_6 (Activation)    (None, 10)                0
=================================================================
Total params: 4,970,698
Trainable params: 4,970,698
Non-trainable params: 0
```

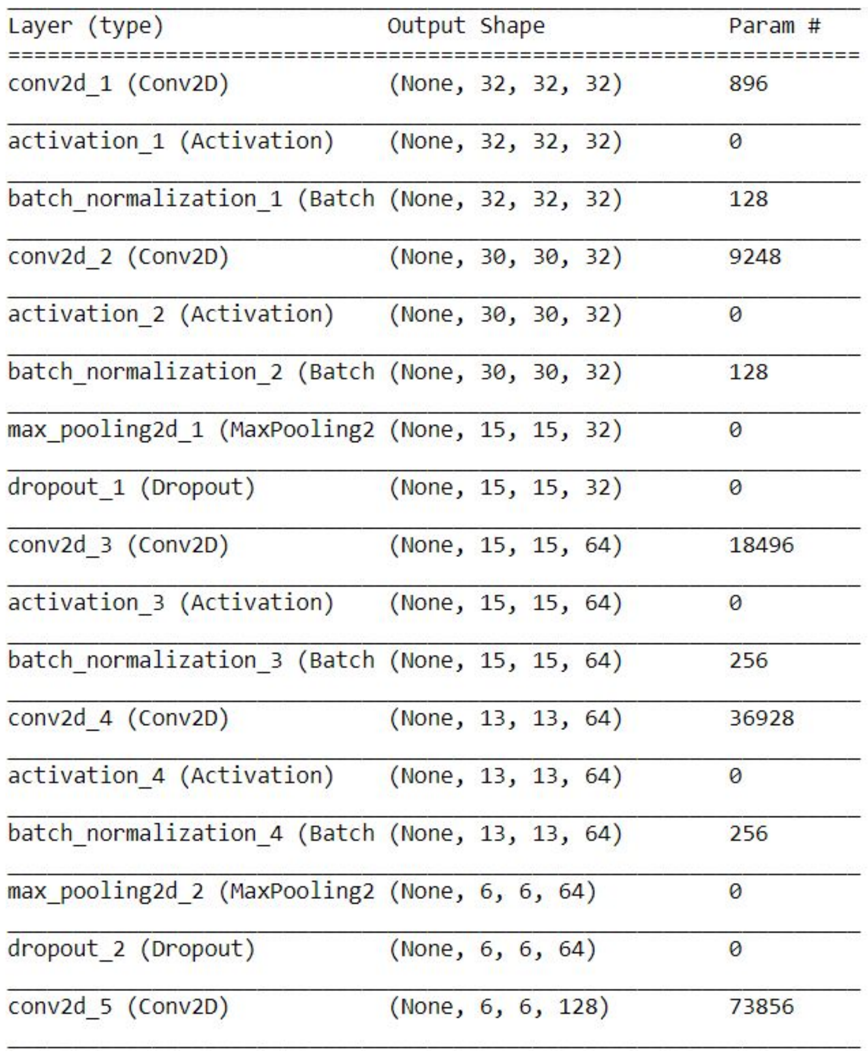
Test loss: 0.60612752037
Test accuracy: 0.8098





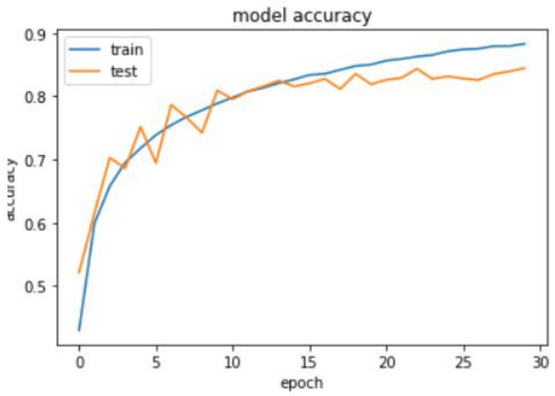
By adding neurons of each “conv2d” layer, we got a better accuracy and less loss. We assume that adding neurons improving the performance of MLP neural network.

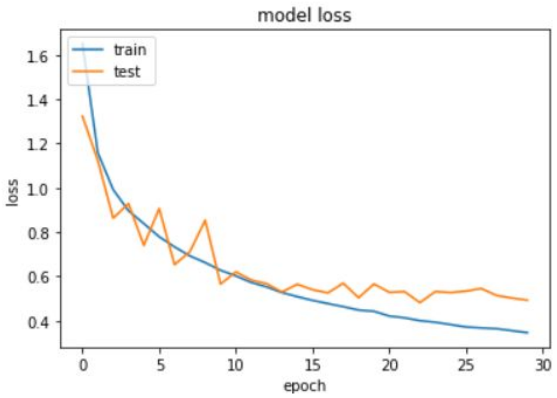
b. Number of layers:
Keeping other parameters the same as above, and adding “conv2d_5”, “conv2d_6”, “max_pooling_2d_3”, “dropout_3” and “batch_normalization” layers after each conv2d layer.


```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 32, 32, 32)        896
_________________________________________________________________
activation_1 (Activation)    (None, 32, 32, 32)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 32, 32, 32)        128
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 30, 30, 32)        9248
_________________________________________________________________
activation_2 (Activation)    (None, 30, 30, 32)        0
_________________________________________________________________
batch_normalization_2 (Batch (None, 30, 30, 32)        128
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 15, 15, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 15, 15, 64)        18496
_________________________________________________________________
activation_3 (Activation)    (None, 15, 15, 64)        0
_________________________________________________________________
batch_normalization_3 (Batch (None, 15, 15, 64)        256
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 13, 13, 64)        36928
_________________________________________________________________
activation_4 (Activation)    (None, 13, 13, 64)        0
_________________________________________________________________
batch_normalization_4 (Batch (None, 13, 13, 64)        256
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
dropout_2 (Dropout)          (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 6, 6, 128)         73856
_________________________________________________________________
```

```
_________________________________________________________________
activation_5 (Activation)    (None, 6, 6, 128)         0
_________________________________________________________________
batch_normalization_5 (Batch (None, 6, 6, 128)         512
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 4, 4, 128)         147584
_________________________________________________________________
activation_6 (Activation)    (None, 4, 4, 128)         0
_________________________________________________________________
batch_normalization_6 (Batch (None, 4, 4, 128)         512
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 2, 2, 128)         0
_________________________________________________________________
dropout_3 (Dropout)          (None, 2, 2, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               262656
_________________________________________________________________
activation_7 (Activation)    (None, 512)               0
_________________________________________________________________
dropout_4 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                5130
_________________________________________________________________
activation_8 (Activation)    (None, 10)                0
=================================================================
Total params: 556,586
Trainable params: 555,690
Non-trainable params: 896
_________________________________________________________________
```

Test loss: 0.492332640553
Test accuracy: 0.8443



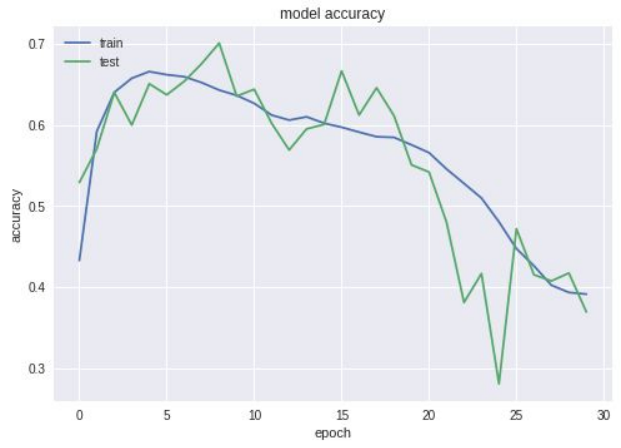


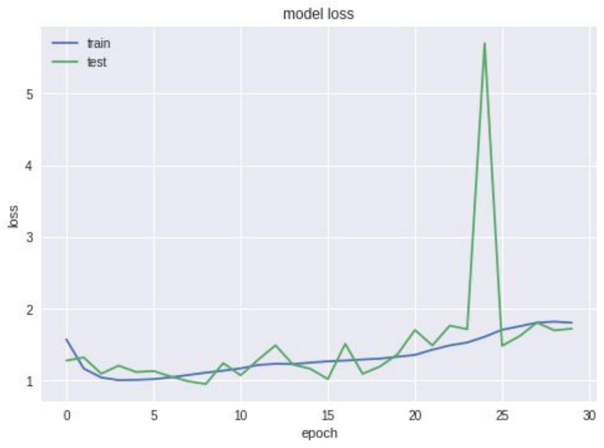
By adding more layers, we got the highest accuracy and the lowest loss; however, the model is kinda overfitting depending on the graphs above because the test loss (val_loss) is above the train loss from the epoch 15 to epoch 30.

4. Learning rate

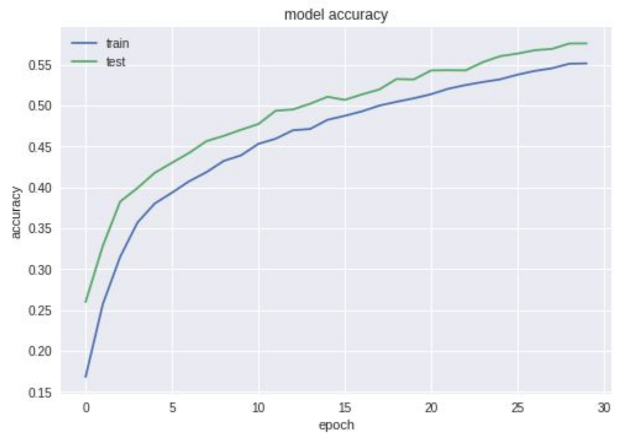
A learning rate defines how much is the model changing the weight of each node according to the loss gradient. With a small learning rate the model may need many epochs to find its limit, for too large a learning rate is going to cause a failure in converging to the minimum loss.

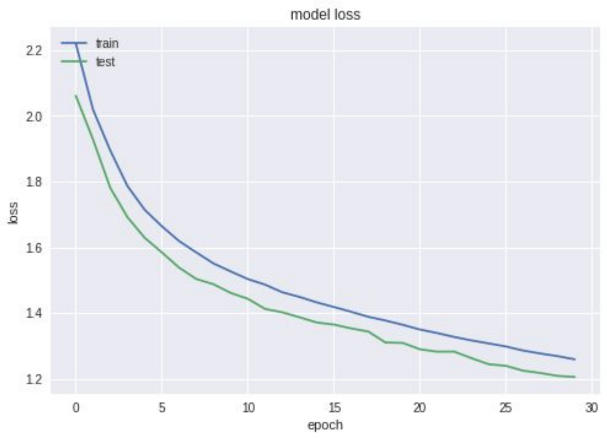
By changing the learning rate from 0.001 to 0.01 we can see if the learning rate is too large, the gradient descent fails to converge and even deverge.





When adjusting the learning rate to a very small value(0.0001), the gradient descent can be very slow. The accuracy is less than learning rate at 0.001 that it may need more epochs to reach the same loss limit.





Overall the best learning rate is 0.001 for this model.

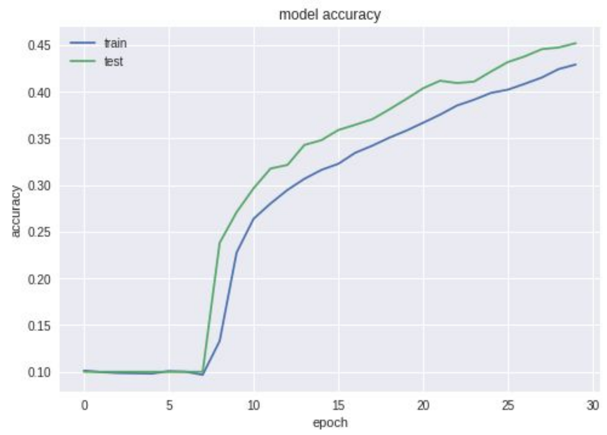
5. Activation function

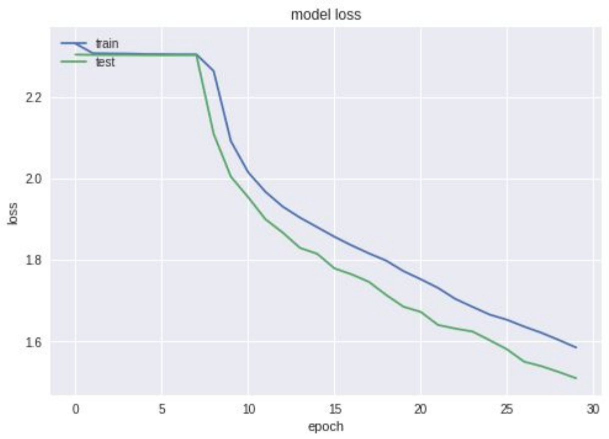
Nodes in the network needs an activation function to trigger it and explain its way to influence the outputs. There are myriad of functions, amongst which there can be suitable ones or not for this specific problem.

The team tried the following functions: sigmoid, tanh, softplus, softmax, elu, selu.

a. sigmoid

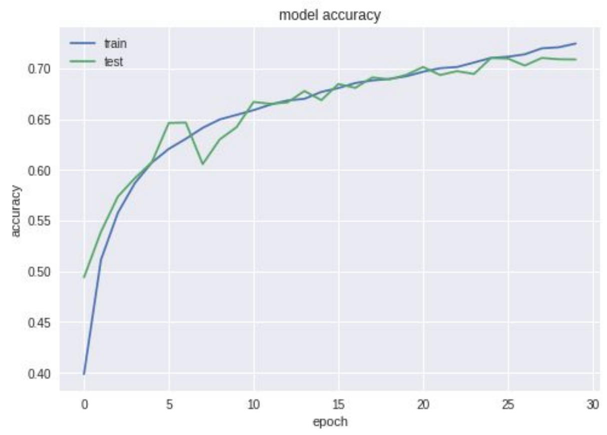
Sigmoid function can compress the output into (0,1). But when implementing backpropagation algorithm sigmoid function can easily cause saturation and gradient loss. After we replaced relu with sigmoid, the accuracy badly decreases and the loss is pretty high.

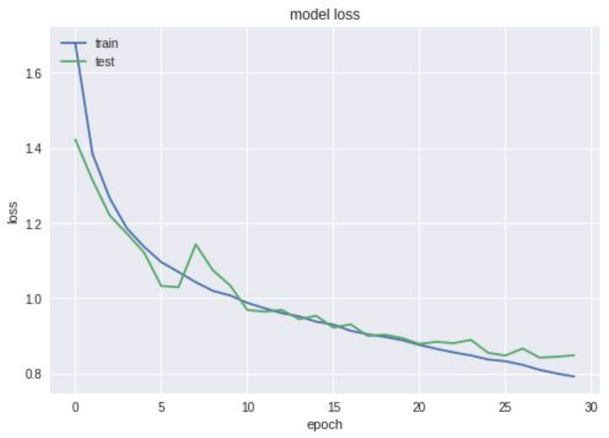




b. tanh

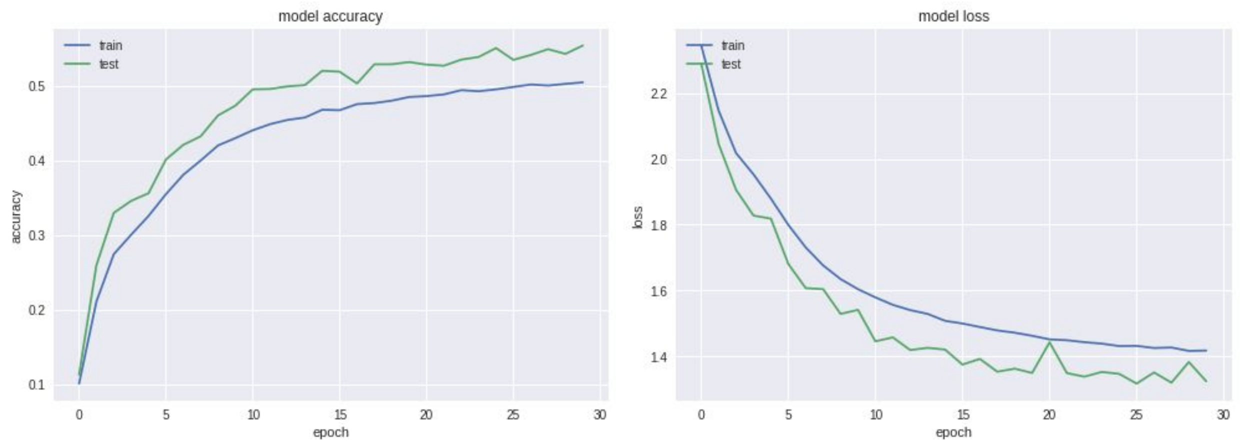
Tanh compresses output into an interval of [-1,1]. Unless sigmoid whose output is always positive, tanh is a zero-centered function, so the weights can be adjusted in all directions. Tanh shows a good performance in accuracy and loss.





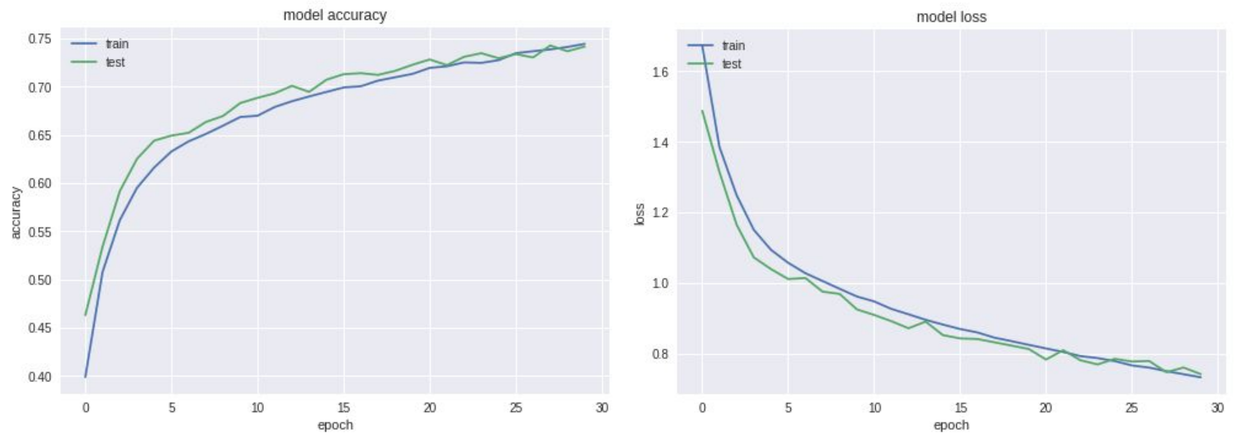
c. softplus

Softplus function is like smooth relu. Unlike relu rejects negative input, softplus can map all real number to positive output. However its derivative is always less than 1, so it has gradient loss problem too. For this model it shows now well with an accuracy just around 0.5.



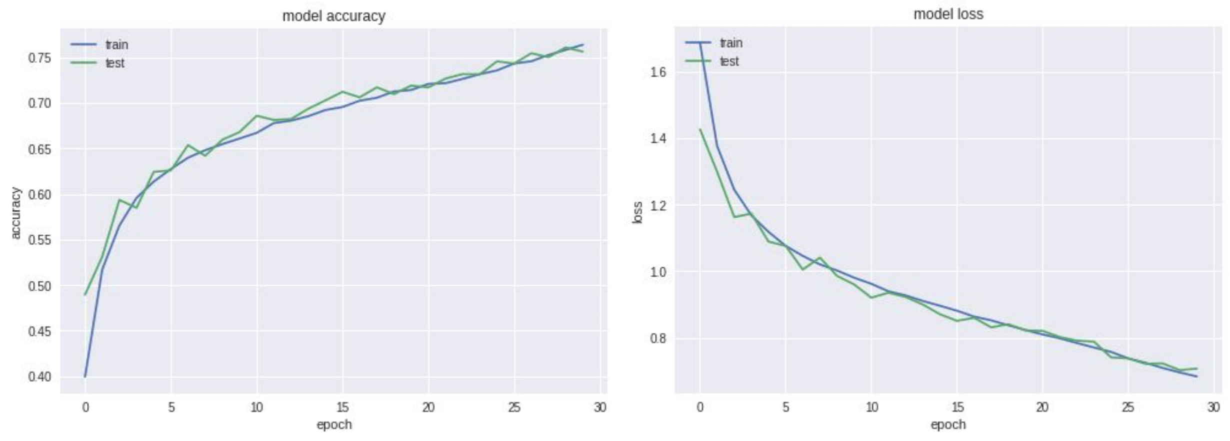
d. softsign

Softsign is kind of like tanh with symmetry at origin and output interval (-1,1). It performs as perfect as relu.



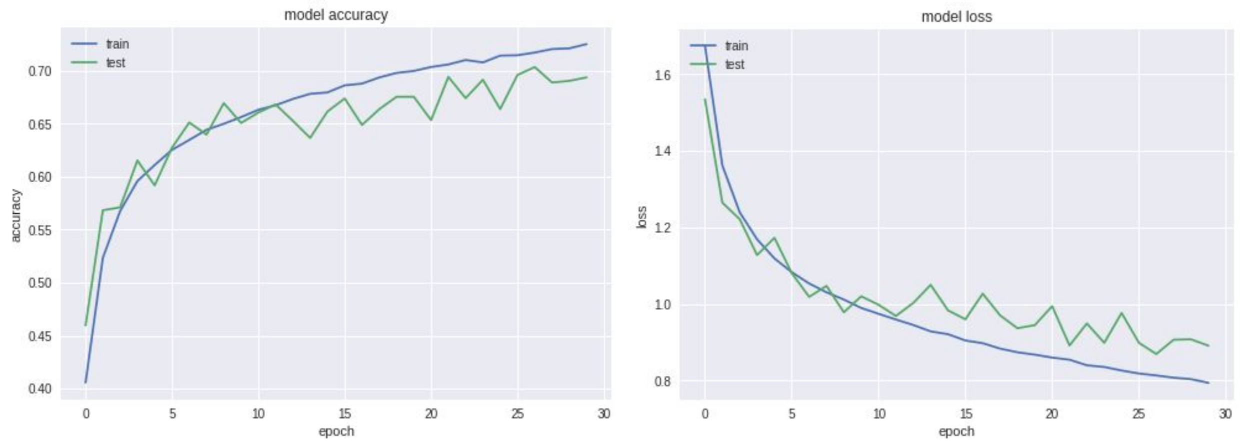
e. elu

Elu function is an enhanced version of relu. It introduces negative input with negative output. Elu brings the highest accuracy among all six tested activation functions.



f. selu

Selu is a new variety of elu. They are similar on negative x half axis, while on positive x half axis the gradient of selu is larger than 1. For this model selu performs well on the training set, but validation set is overfitted.



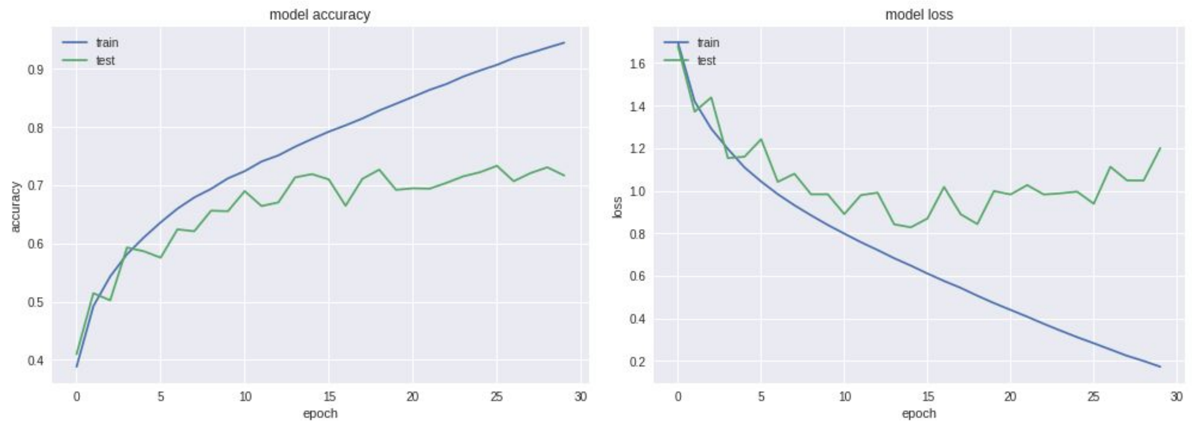
In general, elu and softsign do excellent job in this case, but relu is still the best choice, cause relu is the most simple and efficient activation function without too much complicated computation.

6. Dropout rate

Dropout is another regularization algorithm, that randomly cast a part of neurons out of the net according to a certain rate.

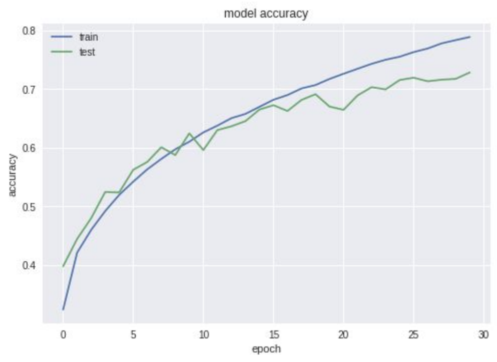
When I set dropout rate as 1…

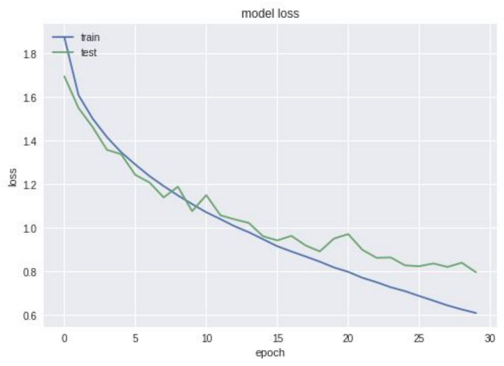
Test loss: 1.8474201045036316
Test accuracy: 0.7192



Nobody can stop the model being overfitted.

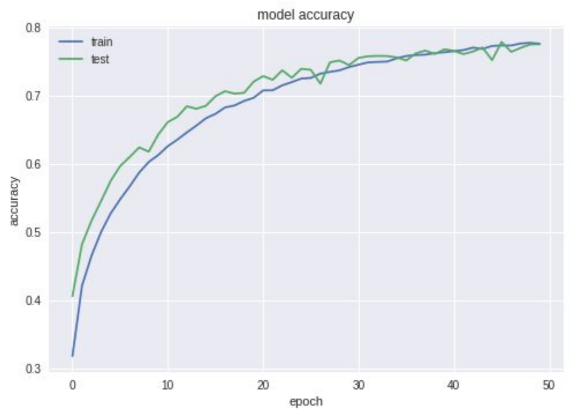
As for a dropout rate of (0.1,0.1,0.2):

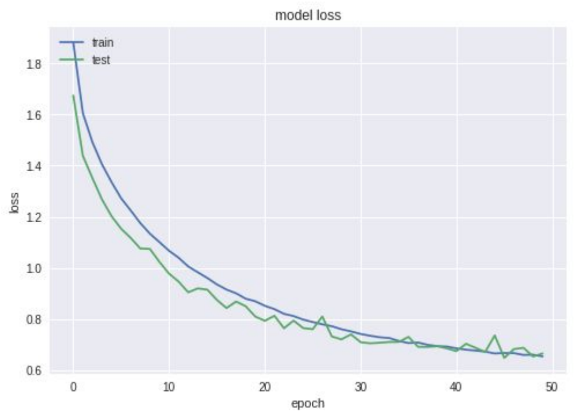




The model is overfitting after epochs.

After trying out different parameters, the team finds a reasonable dropout rate at : (0.25,0.25, 0.5)


model accuracy
train
test
accuracy
epoch


model loss
train
test
loss
epoch