Sojeong Yang
CSC 578 Homework #6 Convolutional Neural Networks
Kaggle username: Soji (Public: 14/0.69015 20:09PM)

- **Description of your best model**

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.Conv2D(32, (3, 3), activation='relu',))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.2))

model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.2))

model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.Dropout(0.2))

model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

The best model improved the better value of accuracy and loss than the initial model. This model has the value of accuracy, 0.7453 and the value of loss, 0.7429. The average value of two trials of the initial model's accuracy is 0.6917 and loss was 0.9268. It shows that the model improved accuracy to 0.0536 and it made loss go down to 0.1839. To build this model, I added two more convolutional layers from the given model, and I changed the number of filters from 64 to 128 in the fifth convolutional layer. It made the value of accuracy higher and value of the loss lower than the initial value. But adding the convolutional layers and adding a number of filters made the running time longer. It makes me think about the computational speed vs. performance trade-off. I got better accuracy from other experiments, but I didn't pick the best model that showed the highest accuracy because I considered how computational speed and running time is also important. The graph below also shows how the model fits pretty well. One of the goals for the best model is minimizing the overfitting and this model is suitable. To minimize the overfitting, I added the dropout after two pooling layers and the last convolutional layers. Finally, the dropout (0.2) with this model minimized the overfitting significantly compared to other experiments.

- **Description of your journey of hyperparameter tuning. Stating from the initial model, describe in depth the hyperparameters and values you tried and the results you got, and your prior expectation and reaction to the results.**

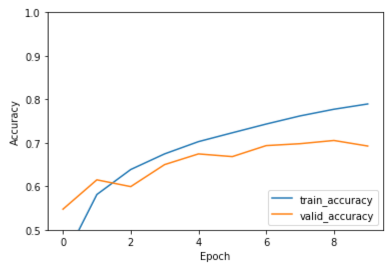
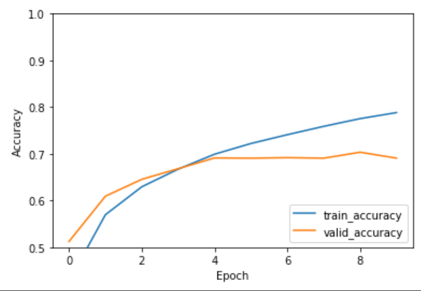
Before I try to tune the model, I tried to understand our initial model. This is the given model.

```
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

Sojeong Yang
CSC 578 Homework #6 Convolutional Neural Networks
Kaggle username: Soji (Public: 14/0.69015 20:09PM)

The input is a 32x32x3 array of pixel values. The input size is 32x32 and the last 3 is for RGB values. This value is also the depth of input and the number is a match with the depth of filter. This model has three convolutional layers. The first convolutional layer has 32 filters and the last two have 64 filters. The kernel/filter size is 3x3 and is using 'relu' for the activation function. After getting through the filter, the size 32x32 of input turns to 30x30 of the feature map. Next to the convolutional layer, the pooling layer reduces the amount of information for each feature obtained from the convolutional layer while preserving the most important information. 'layers.Flatten()' is fully connected input layer and it flattens the output of the previous layer and transforms it into a single vector. After flattening, there are two fully connected layers. The first one takes the inputs from the previous steps and applies weights to them to predict. The second fully connected output layer gives the final probabilities. The model has softmax for the output activation function and it gives probabilities of 10 classes. After I ran with the above model, I got this result:

First trial: 32-64-64	Second trial: 32-64-64

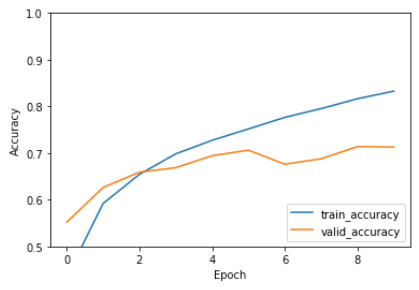
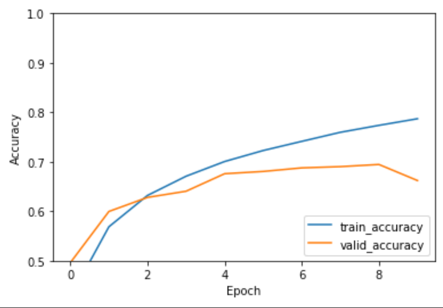
`313/313 - 1s - loss: 0.9129 - accuracy: 0.6926`
`valid_accuracy=0.6926000118255615, valid_loss=0.9128863215446472`

	`313/313 - 1s - loss: 0.9407 - accuracy: 0.6908`
	`valid_accuracy=0.6908000111579895, valid_loss=0.9406942129135132`
	

We can see that it's overfitting between the training set and the validation set and I set the goal to minimize overfitting and get a higher accuracy from the tuned model.

1. Change the number of filters

At first, I changed the number of filters following the instruction suggestions.
I changed the number of filters from 32 to 64 for the first convolutional layer. I expected that by adding more filters and thereby learning more filters would help to get a better accuracy.

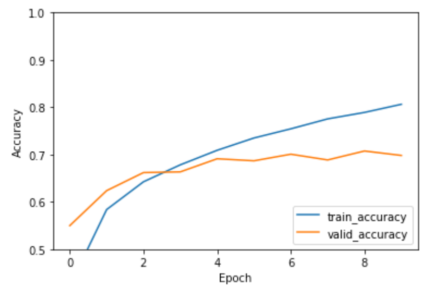
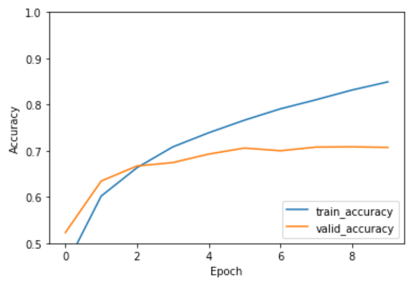
First trial: 64-64-64	Second trial: 64-64-64

`313/313 - 1s - loss: 0.9445 - accuracy: 0.7130`
`valid_accuracy=0.7129999995231628, valid_loss=0.9444859623908997`

	`313/313 - 1s - loss: 1.0412 - accuracy: 0.6621`
	`valid_accuracy=0.6621000170707703, valid_loss=1.0412348508834839`
	

Sojeong Yang
CSC 578 Homework #6 Convolutional Neural Networks
Kaggle username: Soji (Public: 14/0.69015 20:09PM)

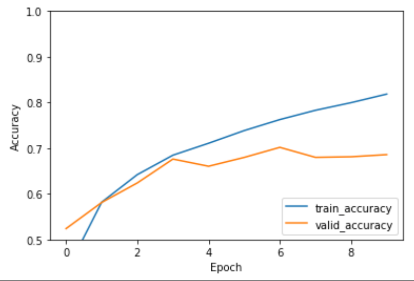
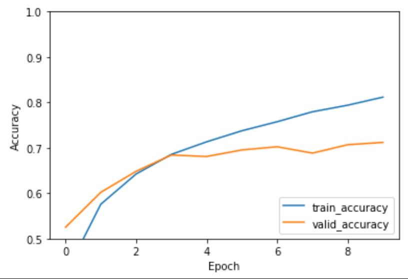
The accuracy of validation got a higher number in the first trial, but in the second trial, the model got worse values of loss and accuracy.

This time, I changed the number of filters from 64 to 128 in the third convolutional layer. In this model the early layer learns fewer convolutional filters while deeper layers in the network will learn more filters and I got this:

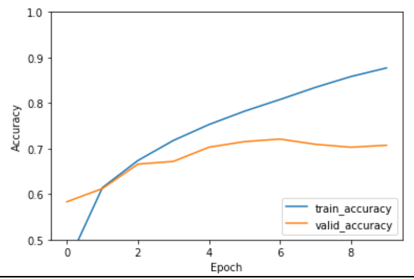
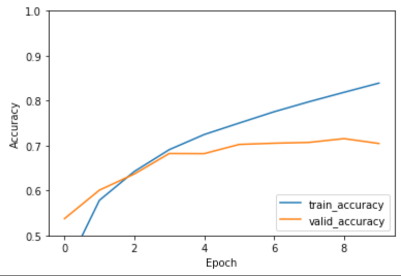
First trial: 32-64-128	Second trial: 32-64-128
`313/313 - 1s - loss: 0.9328 - accuracy: 0.6981` `valid_accuracy=0.6980999708175659, valid_loss=0.9328192472457886` 	`313/313 - 1s - loss: 1.0061 - accuracy: 0.7073` `valid_accuracy=0.7073000073432922, valid_loss=1.0060594081878662` 

Also, it shows better accuracy, but the value of loss and overfitting got bigger. But if I can minimize the overfitting by using the dropout, this model would be good to try.

Next, I tried 64-64-128.

First trial: 64-64-128	Second trial: 64-64-128
`313/313 - 1s - loss: 1.0063 - accuracy: 0.6858` `valid_accuracy=0.6858000159263611, valid_loss=1.006304383277893` 	`313/313 - 1s - loss: 0.8875 - accuracy: 0.7119` `valid_accuracy=0.711899995803833, valid_loss=0.8874683380126953` 

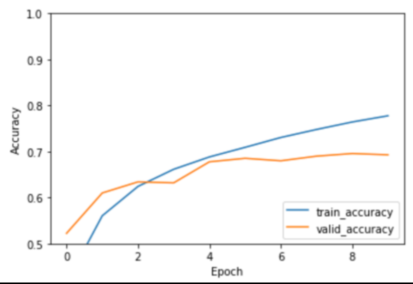
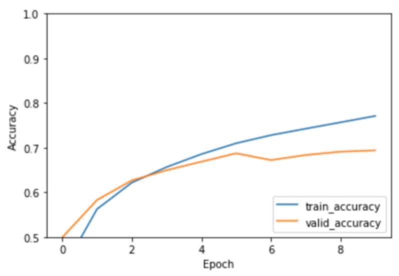
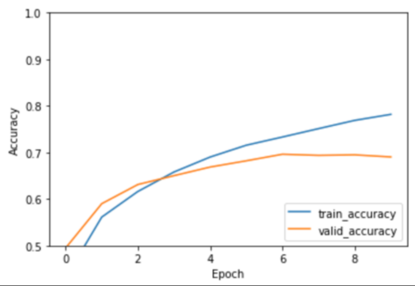
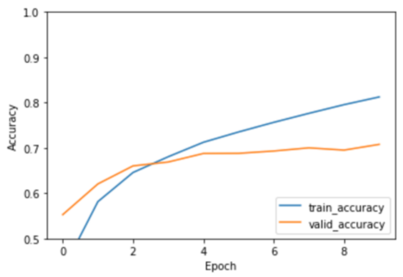
There are big differences between the first trial and second trial. The second trial is a better result than before, but first trial is worse than the initial model.
Next, I tried 64-128-256.

First trial: 64-128-256	Second trial: 64-128-256
`313/313 - 2s - loss: 1.0243 - accuracy: 0.7073` `valid_accuracy=0.7073000073432922, valid_loss=1.024263620376587` 	`313/313 - 2s - loss: 0.9749 - accuracy: 0.7047` `valid_accuracy=0.7046999931335449, valid_loss=0.9749432802200317` 

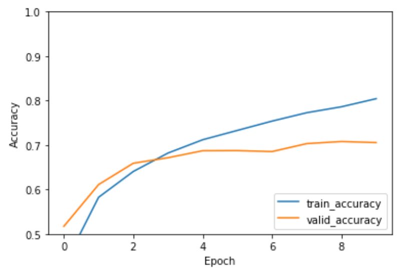
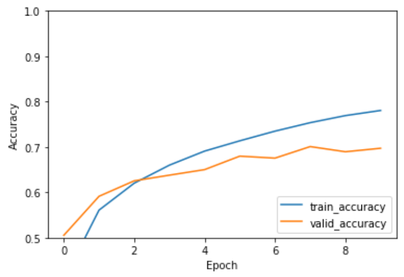
Sojeong Yang
CSC 578 Homework #6 Convolutional Neural Networks
Kaggle username: Soji (Public: 14/0.69015 20:09PM)

It didn't show the dramatic result compared to the trial that I did before and adding more filters increases the time to run the model. Also, validation accuracy is going down when it nears the end.

Next, I modified the number of filters on fully connected layer.

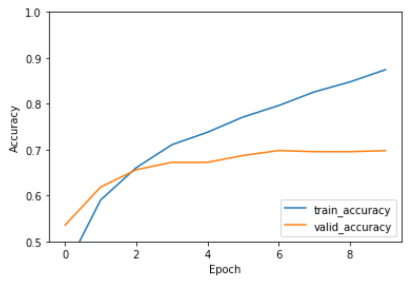
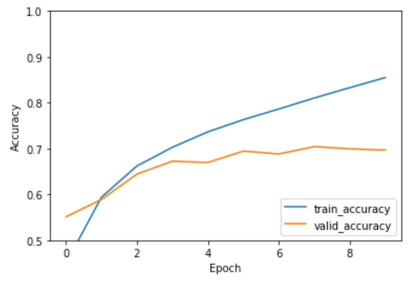
First trial: 32-64-64, 64(given)	Second trial: 32-64-64, 64(given)
`313/313 - 1s - loss: 0.9326 - accuracy: 0.6928` `valid_accuracy=0.692799985408783, valid_loss=0.9326225519180298` 	`313/313 - 1s - loss: 0.9174 - accuracy: 0.6939` `valid_accuracy=0.6938999891281128, valid_loss=0.9174456000328064` 
First trial: 32-64-64, 128(modified)	**Second trial: 32-64-64, 128(modified)**
`313/313 - 1s - loss: 0.9348 - accuracy: 0.6904` `valid_accuracy=0.6904000043869019, valid_loss=0.9348372220993042` 	`313/313 - 1s - loss: 0.9111 - accuracy: 0.7078` `valid_accuracy=0.7077999711036682, valid_loss=0.9110733270645142` 

Compared to the graphs in above, it shows more overfitting when the number of filters got bigger. I wondered if the third convolutional layer got 128 filters, would the accuracy be better or not?

First trial: 32-64-128, 64(given)	Second trial: 32-64-128, 64(given)
`313/313 - 1s - loss: 0.9393 - accuracy: 0.7055` `valid_accuracy=0.7055000066757202, valid_loss=0.9393298029899597` 	`313/313 - 1s - loss: 0.9119 - accuracy: 0.6972` `valid_accuracy=0.6972000002861023, valid_loss=0.9118514657020569` 

Sojeong Yang
CSC 578 Homework #6 Convolutional Neural Networks
Kaggle username: Soji (Public: 14/0.69015 20:09PM)

First trial: 32-64-128, 128(modified)	Second trial: 32-64-128, 128(modified)

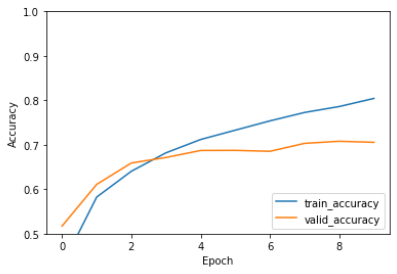
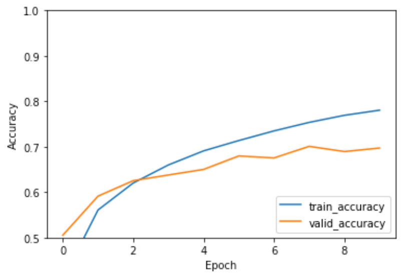
`313/313 - 1s - loss: 1.0815 - accuracy: 0.6978`
`valid_accuracy=0.6977999806404114, valid_loss=1.0815373659133911`

	`313/313 - 1s - loss: 1.0132 - accuracy: 0.6968`
	`valid_accuracy=0.6967999935150146, valid_loss=1.0131886005401611`
	

Compared to the result of 64 and 128, adding more filters makes increases overfitting between training and validation.

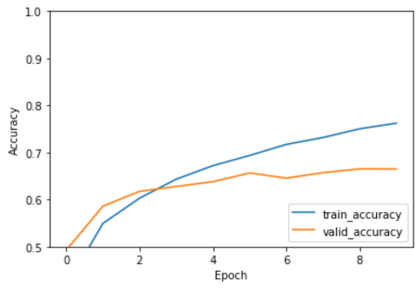
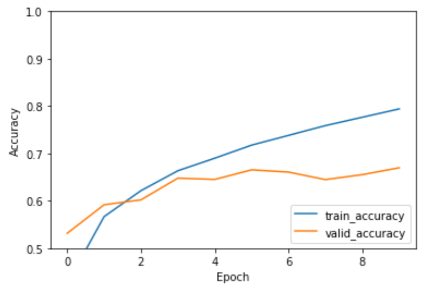
2. Change the size of filters

I experimented with using small filter sizes because it reduces computational costs and weight sharing that ultimately leads to lesser weights for backpropagation. Also, I picked an odd number of sizes because odd-sized filters symmetrically divide the previous layer pixels around the output pixel. Therefore, I experimented with using the filters (3x3 or 5x5).

First trial: filter size(3x3)	Second trial: filter size(3x3)

`313/313 - 1s - loss: 0.9393 - accuracy: 0.7055`
`valid_accuracy=0.7055000066757202, valid_loss=0.9393298029899597`

	`313/313 - 1s - loss: 0.9119 - accuracy: 0.6972`
	`valid_accuracy=0.6972000002861023, valid_loss=0.9118514657020569`
	

First trial: filter size(5x5)	**Second trial: filter size(5x5)**

`313/313 - 1s - loss: 1.0218 - accuracy: 0.6649`
`valid_accuracy=0.664900004863739, valid_loss=1.0218316316604614`

	`313/313 - 1s - loss: 1.0445 - accuracy: 0.6697`
	`valid_accuracy=0.669700026512146, valid_loss=1.044531226158142`
	

Sojeong Yang
CSC 578 Homework #6 Convolutional Neural Networks
Kaggle username: Soji (Public: 14/0.69015 20:09PM)

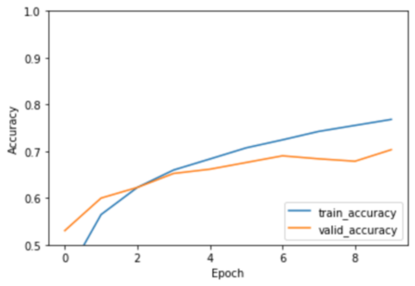
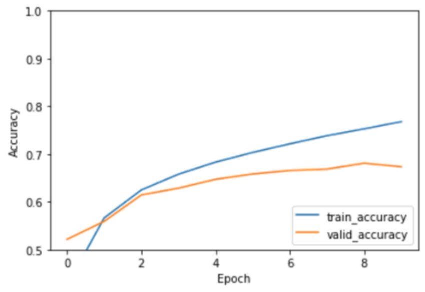
Comparing results of the sizes of filters, the value of loss and accuracy is better at size 3x3 than size 5x5, so I kept using the 3x3 filter.

3. Change the size of stride

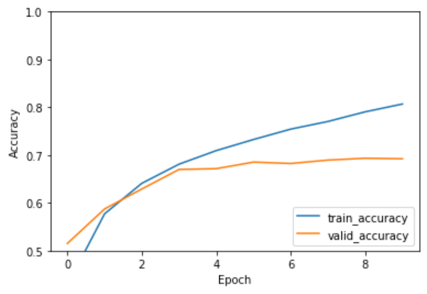
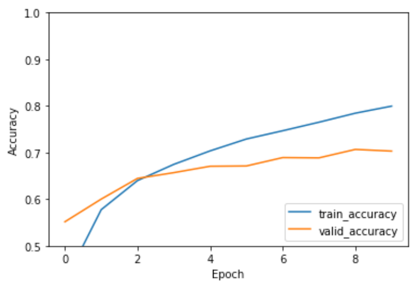
Stride is a parameter of a neural network filter that modifies the amount of movement in the image. If the stride of a neural network is set to 1, the filter moves one pixel or unit at a time like a sliding window. The size of the filter affects the encoded output volume. If the value of strides is bigger, lesser memory is needed for the output and the processing of output is easier because the volume is smaller. Also, it avoids overfitting in the case of image processing having a large number of attributes.

- fully connected layer 128

First trial: 32-64-64, strides = (1,1)	Second trial: 32-64-64, strides = (1,1)

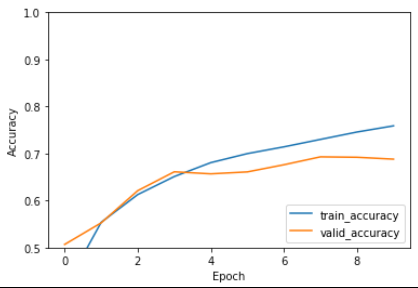
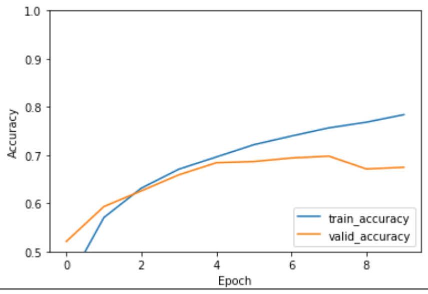
`313/313 - 1s - loss: 0.8923 - accuracy: 0.7033`
`valid_accuracy=0.7032999992370605, valid_loss=0.8922649621963501`

	`313/313 - 1s - loss: 0.9679 - accuracy: 0.6734`
	`valid_accuracy=0.6733999848365784, valid_loss=0.9679328203201294`
	

First trial: 32-64-128, strides = (1,1)	**Second trial: 32-64-128, strides = (1,1)**

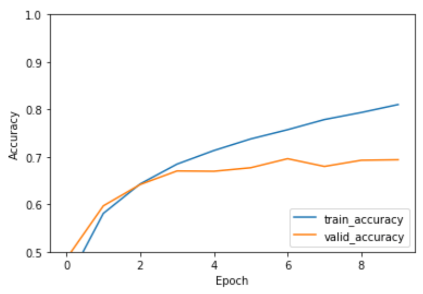
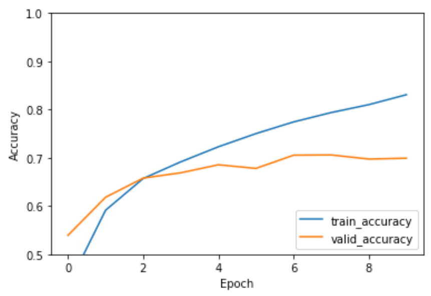
`313/313 - 1s - loss: 0.9610 - accuracy: 0.6923`
`valid_accuracy=0.692300021648407, valid_loss=0.9609550833702087`

	`313/313 - 1s - loss: 0.9223 - accuracy: 0.7031`
	`valid_accuracy=0.7031000256538391, valid_loss=0.922318160533905`
	

- fully connected layer 64

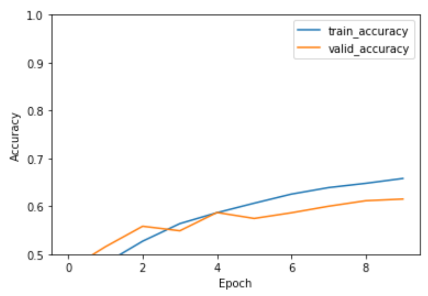
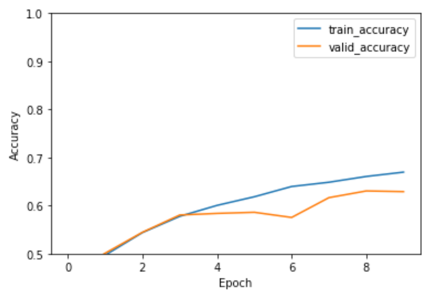
First trial: 32-64-64, strides = (1,1)	Second trial: 32-64-64, strides = (1,1)

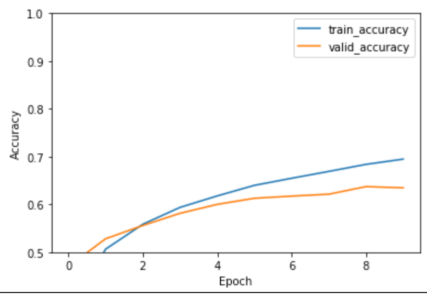
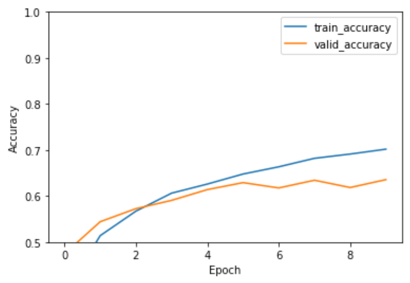
`313/313 - 1s - loss: 0.9307 - accuracy: 0.6880`
`valid_accuracy=0.6880000233650208, valid_loss=0.9306653141975403`

	`313/313 - 1s - loss: 0.9763 - accuracy: 0.6747`
	`valid_accuracy=0.6747000217437744, valid_loss=0.976250290870666 5`
	

Sojeong Yang
CSC 578 Homework #6 Convolutional Neural Networks
Kaggle username: Soji (Public: 14/0.69015 20:09PM)

First trial: 32-64-128, strides = (1,1)	Second trial: 32-64-128, strides = (1,1)
`313/313 - 1s - loss: 0.9387 - accuracy: 0.6940` `valid_accuracy=0.6940000057220459, valid_loss=0.9387248754501343` 	`313/313 - 1s - loss: 0.9500 - accuracy: 0.6991` `valid_accuracy=0.6991000175476074, valid_loss=0.949971079826355` 

Compared to strides = (1,1) with different number of filters in convolutional layers, 32-64-128 is overfitted more but it has better accuracy. If I change the other hyperparameters so it can minimize the overfitting, it has a chance to become the better model.

First trial: 32-64-64, strides = (2,2)	Second trial: 32-64-64, strides = (2,2)
`313/313 - 0s - loss: 1.1054 - accuracy: 0.6151` `valid_accuracy=0.6151000261306763, valid_loss=1.1053822040557861` 	`313/313 - 0s - loss: 1.0273 - accuracy: 0.6290` `valid_accuracy=0.6290000081062317, valid_loss=1.0272955894470215` 

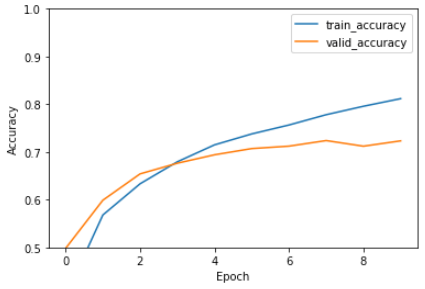
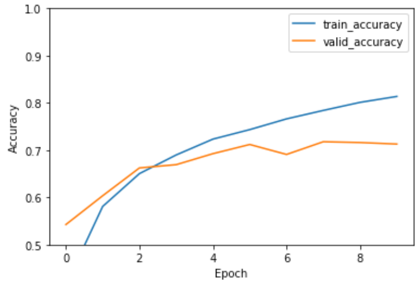
First trial: 32-64-128, strides = (2,2)	Second trial: 32-64-128, strides = (2,2)
`313/313 - 0s - loss: 1.0656 - accuracy: 0.6346` `valid_accuracy=0.6345999836921692, valid_loss=1.0655791759490967` 	`313/313 - 0s - loss: 1.0672 - accuracy: 0.6356` `valid_accuracy=0.6355999708175659, valid_loss=1.0672247409820557` 

Using 'strides = (2,2)' made a lower accuracy. I expected that the kernel/filter slides to the next by 2 so that might cause more loss of information than 'strides = (1,1)'. The good thing about using ‘strides = (2,2)’ is that the model runs much faster than ‘strides = (1,1)’. I can save more time in getting results.

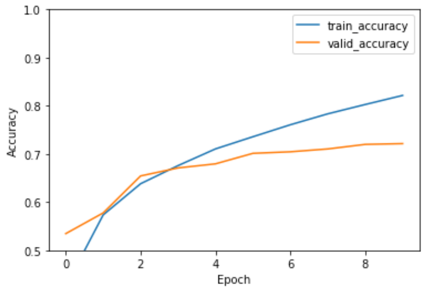
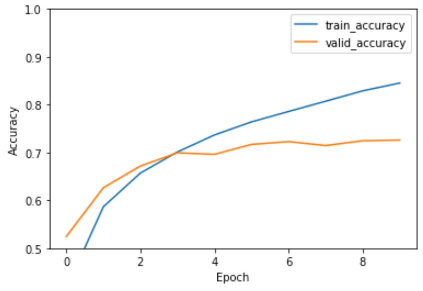
4. Add more convolution layers

Add more convolutional layer, without strides

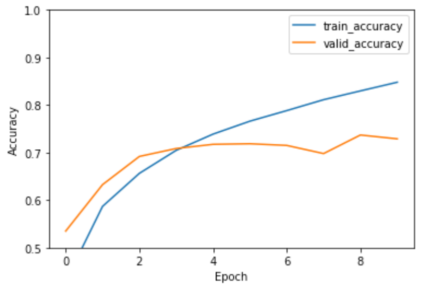
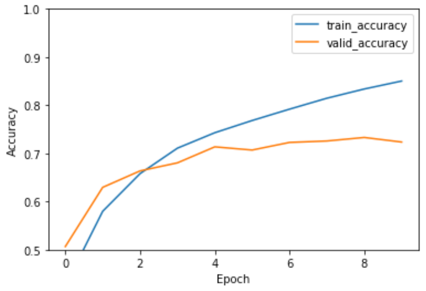
First trial: 32-32-64-64-64	Second trial: 32-32-64-64-64

`313/313 - 2s - loss: 0.8489 - accuracy: 0.7233`
`valid_accuracy=0.7232999801635742, valid_loss=0.8488875031471252`

	`313/313 - 2s - loss: 0.8720 - accuracy: 0.7129`
	`valid_accuracy=0.7128999829292297, valid_loss=0.8720118403434753`
	

First trial: 32-32-64-64-128	Second trial: 32-32-64-64-128

`313/313 - 2s - loss: 0.8917 - accuracy: 0.7215`
`valid_accuracy=0.7214999794960022, valid_loss=0.8916969895362854`

	`313/313 - 2s - loss: 0.8727 - accuracy: 0.7260`
	`valid_accuracy=0.7260000109672546, valid_loss=0.8726717829704285`
	

First trial: 32-32-64-64-256	Second trial: 32-32-64-64-256

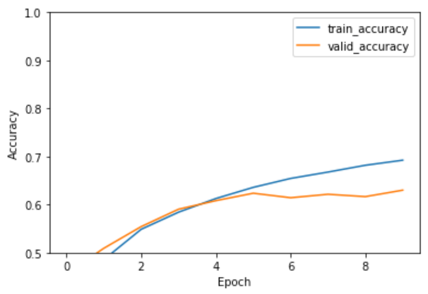
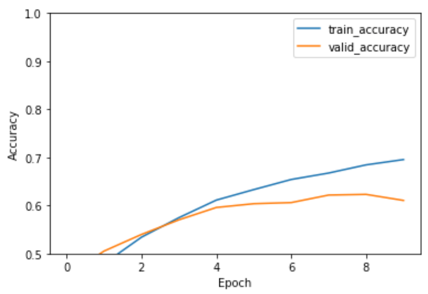
`313/313 - 2s - loss: 0.8591 - accuracy: 0.7289`
`valid_accuracy=0.7289000153541565, valid_loss=0.8591293096542358`

	`313/313 - 2s - loss: 0.8900 - accuracy: 0.7236`
	`valid_accuracy=0.7235999703407288, valid_loss=0.8900344967842102`
	

According to the graph in above, adding more convolutional layers help to get higher accuracy. I expected that if I add more filters, it could take more running time and the result could show bigger overfitting. The graphs show exactly what I expected.

Sojeong Yang
CSC 578 Homework #6 Convolutional Neural Networks
Kaggle username: Soji (Public: 14/0.69015 20:09PM)

Add more convolutional layer, 128 fully and strides

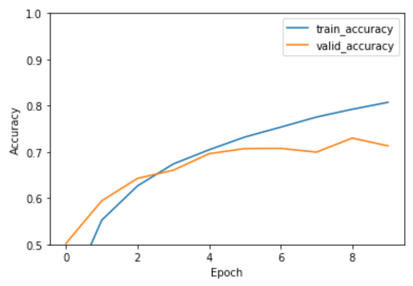
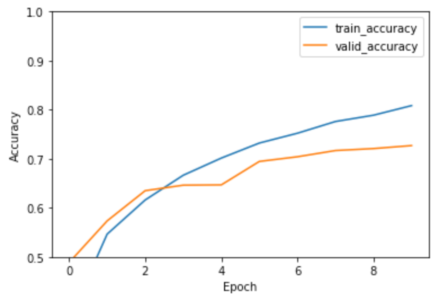
First trial: 32-64-128, strides = (2,2)	Second trial: 32-64-128, strides = (2,2)

`313/313 - 0s - loss: 1.0665 - accuracy: 0.6300`
`valid_accuracy=0.6299999952316284, valid_loss=1.0665456056594849`

	`313/313 - 0s - loss: 1.1327 - accuracy: 0.6105`
	`valid_accuracy=0.6104999780654907, valid_loss=1.1326944828033447`
	

Adding convolutional layers slowed running speed so I tried to add ‘stride = (2,2)’ because I got better running time the former experiment. But it made the value of the accuracy worse, as well as value of loss.

5. Add more Fully Connected layers.

32-32-64-64-64, 64-64(add 1)	32-32-64-64-64, 64-64-64(add 2)

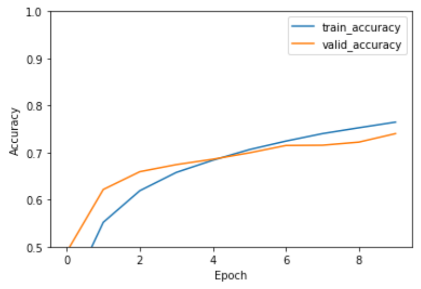
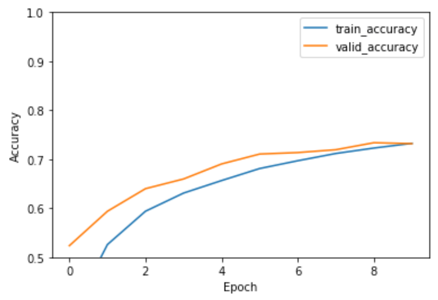
`313/313 - 2s - loss: 0.8752 - accuracy: 0.7131`
`valid_accuracy=0.713100016117096, valid_loss=0.8752285242080688`

	`313/313 - 2s - loss: 0.8366 - accuracy: 0.7269`
	`valid_accuracy=0.7268999814987183, valid_loss=0.8365650177001953`
	

Adding more fully connected layers didn’t show the significant difference and it made the program slow.

6. Adding dropout layers, adding regularization and removing layers

No stride

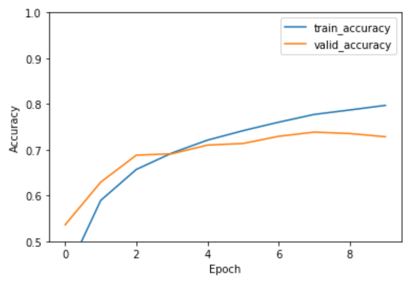
32-32-64-64-64-64 + add dropout (0.1)	32-32-64-64-64-64 + add dropout (0.2)

`313/313 - 2s - loss: 0.7513 - accuracy: 0.7403`
`valid_accuracy=0.7402999997138977, valid_loss=0.7513081431388855`

	`313/313 - 2s - loss: 0.7758 - accuracy: 0.7318`
	`valid_accuracy=0.7318000197410583, valid_loss=0.7757869362831116`
	

Sojeong Yang
CSC 578 Homework #6 Convolutional Neural Networks
Kaggle username: Soji (Public: 14/0.69015 20:09PM)

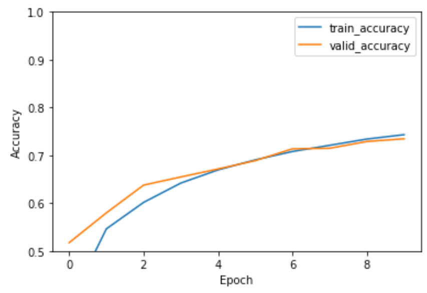
32-32-64-64-128-64 + add dropout (0.1)

```
313/313 - 2s - loss: 0.8084 - accuracy: 0.7285
valid_accuracy=0.7285000085830688, valid_loss=0.8084425926208496
```



32-32-64-64-128-64 + add dropout (0.2)

```
313/313 - 2s - loss: 0.7773 - accuracy: 0.7346
valid_accuracy=0.7346000075340271, valid_loss=0.7772950530052185
```

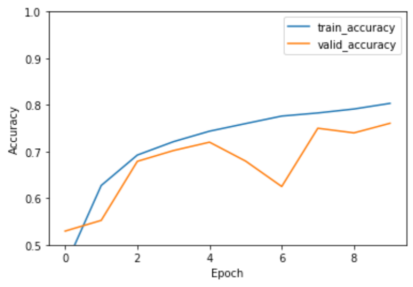


Adding the dropout made a significant difference to minimize overfitting. I expected that dropout is one of the hyperparameter to reduce the overfitting. It worked very well with more convolutional layers.

Next, I tried to use regularization.

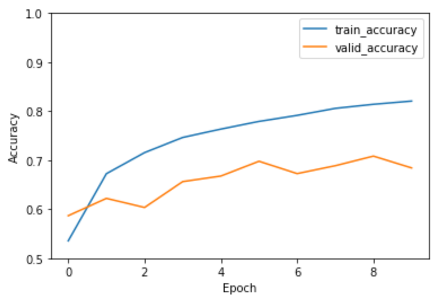
32-32-64-64-128-64 + dropout (0.2)122s

```
313/313 - 4s - loss: 0.9481 - accuracy: 0.7606
valid_accuracy=0.7605999708175659, valid_loss=0.948124885559082
```



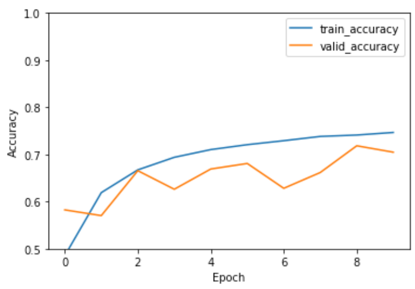
Simple, 32-64-64-64 + no dropout (45s)

```
313/313 - 2s - loss: 1.3233 - accuracy: 0.6842
valid_accuracy=0.6841999888420105, valid_loss=1.3233110904693604
```



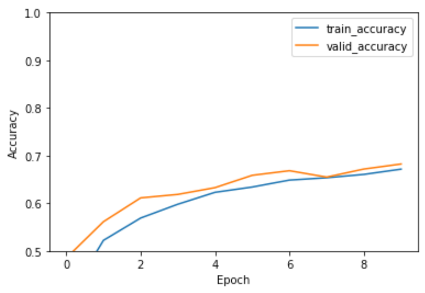
Simple, 32-64-64-64 + dropout (0.2), 48s

```
313/313 - 2s - loss: 1.1959 - accuracy: 0.7048
valid_accuracy=0.704800009727478, valid_loss=1.1959480047225952
```



32-64-64-64+ dropout (0.2), (36s)

```
313/313 - 1s - loss: 1.1134 - accuracy: 0.6824
valid_accuracy=0.6823999881744385, valid_loss=1.1133841276168823
```



Using regularization helped to minimize overfitting but using it by itself still has overfitting and it works better it with the dropout.

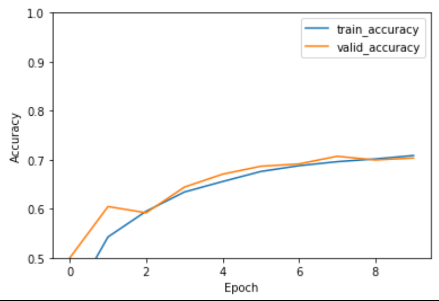
Sojeong Yang

CSC 578 Homework #6 Convolutional Neural Networks

Kaggle username: Soji (Public: 14/0.69015 20:09PM)

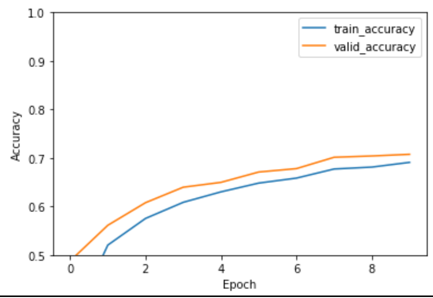
32-32-64-64-64 + dropout (0.2), 75s

```
313/313 - 2s - loss: 1.1034 - accuracy: 0.7035
valid_accuracy=0.703499972820282, valid_loss=1.103441834449768
```



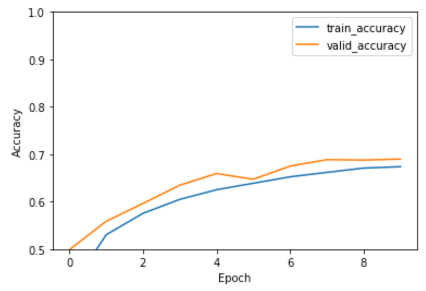
32-32-64-64-64-64 + dropout (0.2), 100s

```
313/313 - 4s - loss: 1.0421 - accuracy: 0.7075
valid_accuracy=0.7074999809265137, valid_loss=1.0421394109725952
```



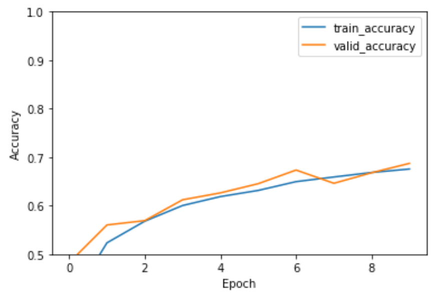
32-64-64-128 + dropout (0.2), 38s

```
313/313 - 1s - loss: 1.1182 - accuracy: 0.6898
valid_accuracy=0.6898000240325928, valid_loss=1.1182128190994263
```



32-64-128-128 + dropout (0.2), 48s

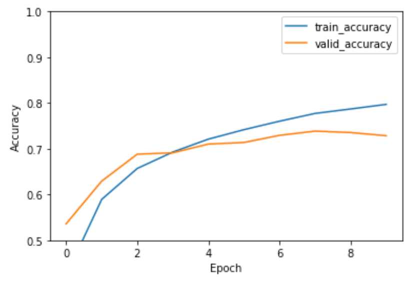
```
313/313 - 1s - loss: 1.1255 - accuracy: 0.6872
valid_accuracy=0.6872000098228455, valid_loss=1.1255100965499878
```



No regularization

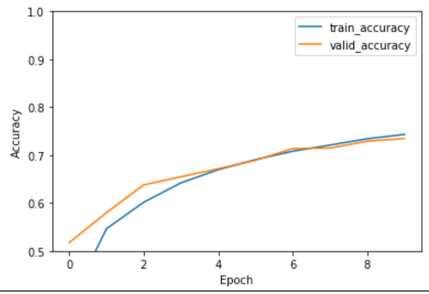
32-32-64-64-128-64 + add dropout (0.1)

```
313/313 - 2s - loss: 0.8084 - accuracy: 0.7285
valid_accuracy=0.7285000085830688, valid_loss=0.8084425926208496
```



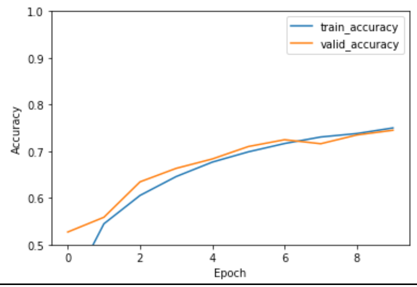
32-32-64-64-128-64 + add dropout (0.2)

```
313/313 - 2s - loss: 0.7773 - accuracy: 0.7346
valid_accuracy=0.7346000075340271, valid_loss=0.7772950530052185
```



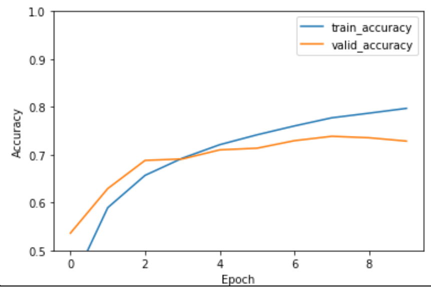
32-32-64-64-128-64 + add dropout (0.2)

```
313/313 - 2s - loss: 0.7444 - accuracy: 0.7450
valid_accuracy=0.7450000047683716, valid_loss=0.7443677186965942
```



32-32-64-64-64-256 + add dropout (0.1)

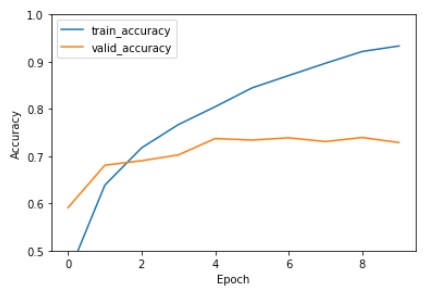
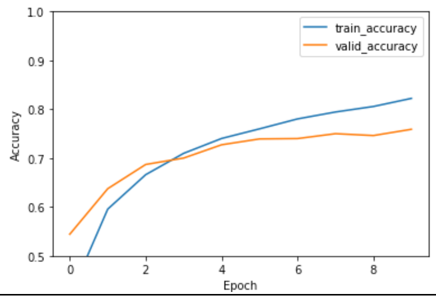
```
313/313 - 2s - loss: 0.8084 - accuracy: 0.7285
valid_accuracy=0.7285000085830688, valid_loss=0.8084425926208496
```



I was so surprised about the above results. The overfitting problem fixed, and the accuracy of training set and valid set ended similar values. I tried one more experiment with same hyperparameters and I still got similar results.

Add stride 1

32-32-64-64-128-64 + no dropout, 98s	32-32-64-64-128-64 + dropout (0.2), 100s

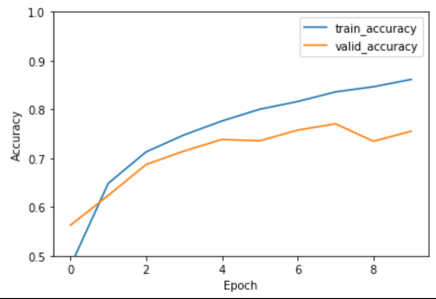
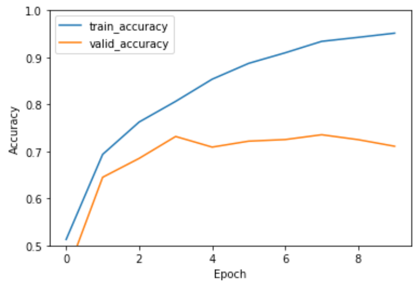
`313/313 - 3s - loss: 1.1135 - accuracy: 0.7289`
`valid_accuracy=0.7289000153541565, valid_loss=1.1134588718414307`

	`313/313 - 3s - loss: 0.7294 - accuracy: 0.7590`
	`valid_accuracy=0.7590000033378601, valid_loss=0.7293640971183777`
	

I added the 'stride = (1,1)' to the above setting that included the dropout and the running time got longer but it got better accuracy and loss.

7. Add batchNormalization layers

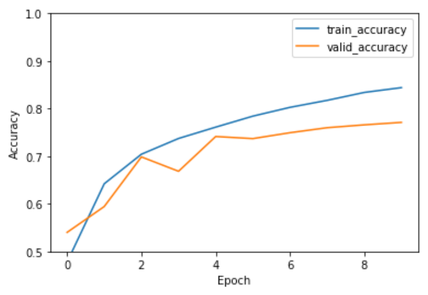
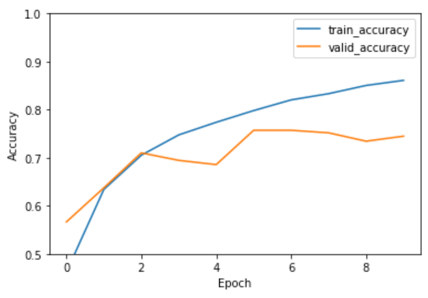
In batch normalization, the process of adjusting the mean and variance is not a separate process, but is included in the neural network, and the process of adjusting the mean and variance during training is controlled together. It helps to reduce the gradient vanishing problems.

32-32-64-64-128-64 + dropout (0.2), 114s	32-32-64-64-128-64 + no dropout, 104s

`313/313 - 5s - loss: 0.7943 - accuracy: 0.7553`
`valid_accuracy=0.755299985408783, valid_loss=0.7942706346511841`

	`313/313 - 4s - loss: 1.3901 - accuracy: 0.7110`
	`valid_accuracy=0.7110000252723694, valid_loss=1.3900779485702515`
	

The dropout fixed the overfitting significantly in these two graphs above. Adding the batch-normalization layers didn't show a big difference.

Sojeong Yang
CSC 578 Homework #6 Convolutional Neural Networks
Kaggle username: Soji (Public: 14/0.69015 20:09PM)

32-32-64-64-128-128 + dropout (0.2/0.3/0.3) + stride (1,1) + no regularization + batchNormalization, 130s	32-32-64-64-128-64 + dropout (0.2/0.2/0.2) + stride (1,1) + no regularization + batchNormalization, 130s
`313/313 - 4s - loss: 0.7132 - accuracy: 0.7711` `valid_accuracy=0.7710999846458435, valid_loss=0.7131696343421936` 	`313/313 - 4s - loss: 0.8419 - accuracy: 0.7448` `valid_accuracy=0.7447999715805054, valid_loss=0.8418536186218262` 

According to the left graph in above, I got the best accuracy of the entire experiments, but it also took the longest running time. The overfitting is still existing, but it is lower than the right graph.

- Include in the journey description, pick one non-best model that you caught your attention or want to put forward and describe the model and discuss its performance results.

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), strides=(1,1), padding='same', activation='relu',
input_shape=(32, 32, 3)))
model.add(layers.Conv2D(32, (3, 3), strides=(1,1), padding='same', activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.2))

model.add(layers.Conv2D(64, (3, 3), strides=(1,1), padding='same', activation='relu'))
model.add(layers.Conv2D(64, (3, 3), strides=(1,1), padding='same', activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.3))

model.add(layers.Conv2D(128, (3, 3), strides=(1,1), padding='same',
activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.Dropout(0.3))

model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

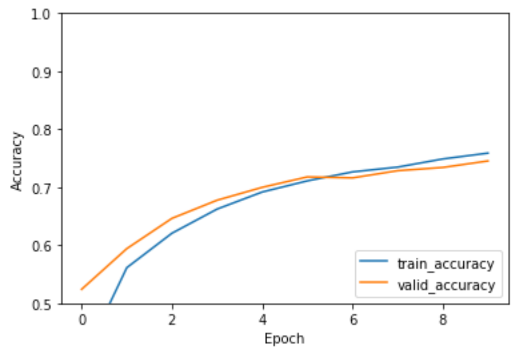
I didn't pick this model to the best model but I'd like to talk about the model. I added the 'strides = (1,1)' and 'BatchNormalization()' in the best model that I picked. It showed the highest accuracy and the lowest loss. The problem was the computational speed. 'BatchNormalization()' helped in getting higher accuracy and reduced the error rate but I can't make a better speed than compared to the others. Also, I expected it to give a similar effect as the dropout. But when I didn't add the dropout, it didn't minimize the overfitting. I think that if I do more experiments about batch-normalization, I might find a better model.

- **Your final conclusion on the best model**

The best model shows less overfitting compared to other experiments. Minimizing the overfitting is an important part between the training and validation. Because less overfitting means this model is generalized well. Also, in other models which had better accuracy, the running time averaged 130s but the running time of this model was average 66.82s. I think that the speed is also very important. Overall, the best model has less overfitting, less value of loss and a faster running time.

The best model

```
313/313 - 2s - loss: 0.7429 - accuracy: 0.7453
valid_accuracy=0.7452999949455261, valid_loss=0.7428522706031799
```



- **Your reaction and reflection on this assignment overall.**

I used to Kaggle several times to find the dataset, but I've never participated in a competition. Competing on performance with others gave me more motivation to do this assignment. Initially, I expected that I could improve the accuracy more than 10% but it was not that easy to make this true. Sometimes, the results were worse than my initial expectations of values. If I got better accuracy, it took so much time to get the result, so it made me frustrated to wait. It's challenging to compromise on all hyperparameters and make the best model. I tried to use 'ImageDataGenerator' and it improved the running time, but I didn't get good accuracy or a better loss this time. I wonder if I tried 'ImageDataGenerator' with the other hyperparameters, I might get a better model that runs faster, has better accuracy and has less overfitting. It was a fun journey to find the best model and it also gave me time to think about the trade-off and hyperparameters compromise.