Project3

CNN

Dataset: Derma MNIST

Zahra Aynedast

Starting:

Finding best model

Step 0:

I started with a simple model with **VGG16 baseline and just two blocks:**

Model: two block VGG16 baseline

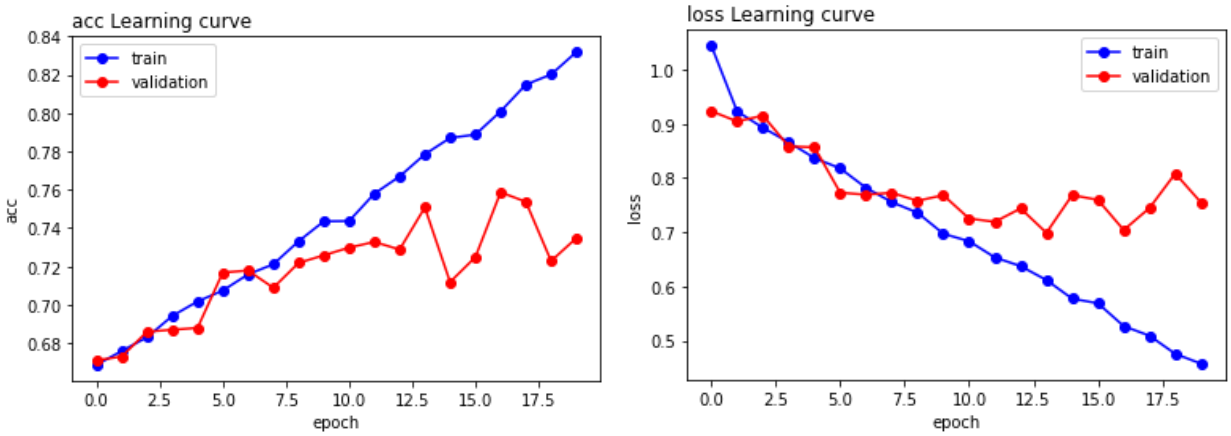
Preprocessing: 1 0

optimizer='Adam'

loss='categorical crossentropy'

batch size=32, epochs=20,

```
model = Sequential ()
model.add (Conv2D (32, (3, 3), activation='relu', padding='same', input_shape= (28, 28, 3)))
                    model.add (Conv2D (32, (3, 3), activation='relu', padding='same'))
model.add (MaxPool2D ((2, 2), strides= (2, 2)))


model.add (Conv2D (64, (3, 3), activation='relu', padding='same'))
model.add (Conv2D (64, (3, 3), activation='relu', padding='same'))
model.add (MaxPool2D ((2, 2), strides= (2, 2)))


model.add (Flatten ())
model.add (Dense (7, activation='softmax'))
```



We got **best acc at epoch 17 = 0.75**

But generally, the learning curve shows **overfitting**, and the distance between train and validation is **high**.

The slope of blue chart is not horizontal yet and it shows that we can have better result with more epochs.

But for validation the acc is going to be the same and the loss will increase.

Conclusion:

The score is not good enough, so we will **increase** the **network capacity.**

20 epoch is enough for now. After reaching a satisfying model, we will test more epochs.

Mabey we should **test smaller learning rates** too to decrease jumping in validation chart.

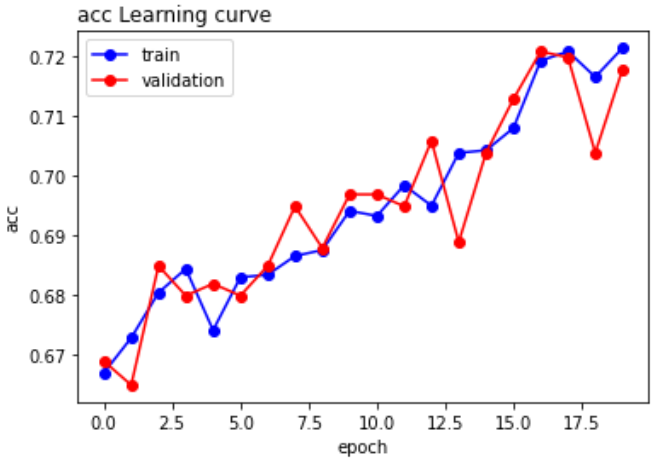
Step 1:

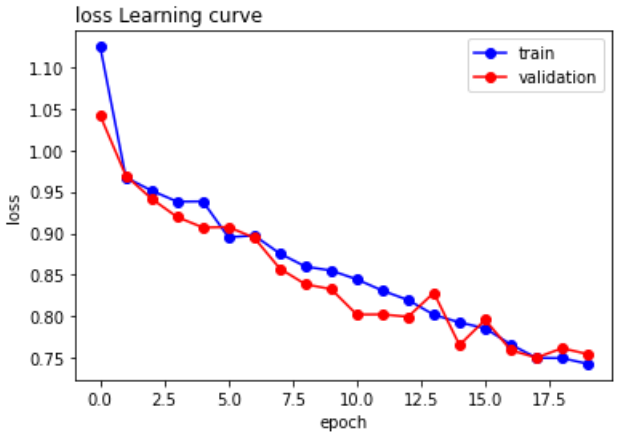
Our model is still the same with some changes:

I added **two VGG16 baseline blocks** to increase network capacity

And **added a dropout** layer with 0.2, in every block to reduce the overfitting.

Model: Four VGG16 block





We got best acc at epoch 17 = 0.7178

According to our diagrams, the overfitting problem is fitted now and we have good charts.

The distance between diagrams is really small, we have no overfittings and no underfitting.

The slope in both diagrams for both training data and validation data is not horizontal yet and it shows that with more epochs we may have better results.

*I tested adding another dense layer with 100 unit before last layer , but result didn't changes that much

Conclusion:

With adding **dropout** layer, **overfitting** problem **was fitted**.

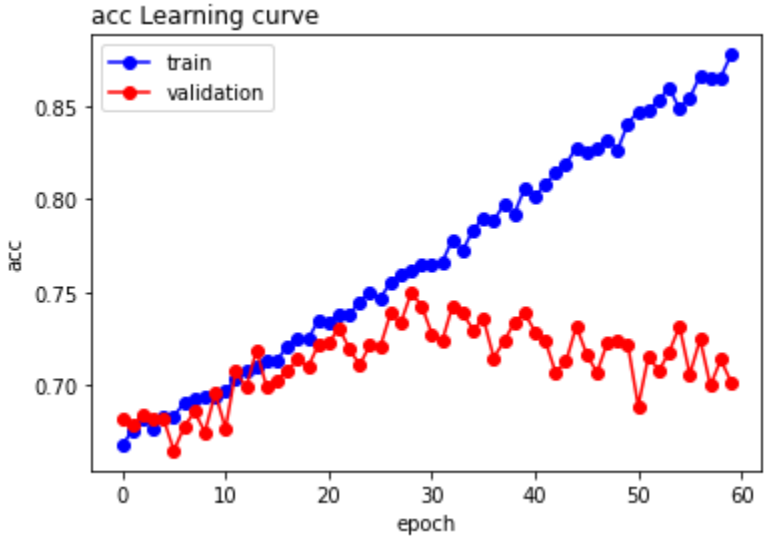
20 epochs are **not enough** for training.

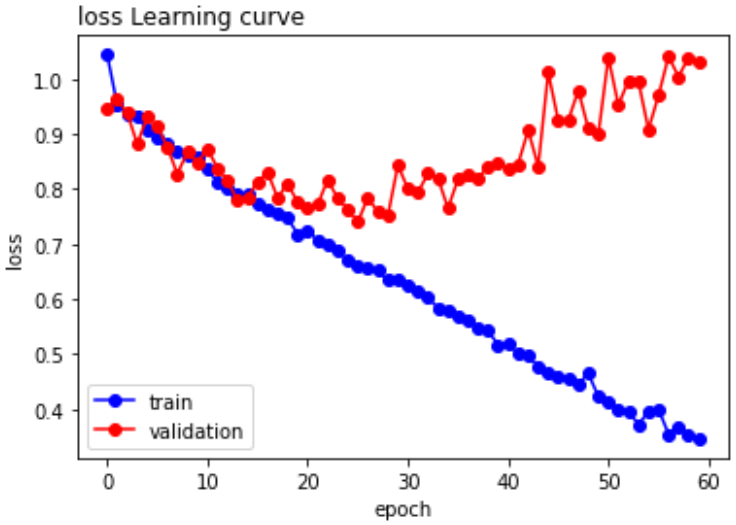
The **acc**, is **not that good** and we should try to make it better.

We still have **jumping in acc** learning diagram.

Step 2:

In this step we **increased** our epoch from **20 to 60**, to give more time for training:





We got **best result at epoch 29 = 0.7498**

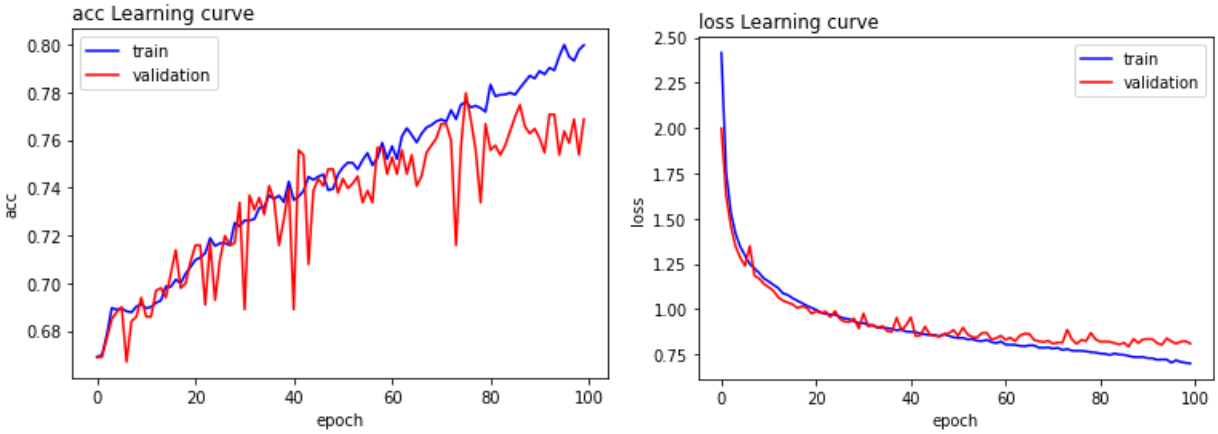
This much epochs were enough for training data (we got 0.89 acc in training) but the result didn't change for validation data and overfitting happened.

Conclusion:

30 epochs are the best for validation acc. So, we should try preventing overfitting while increasing epochs, or we should use just 30 epochs.

Step 3:

In this step, I increased epochs from 60 to 100, and added a **l2 regularization** to prevent overfitting. I also changed learning rate to **0.0001** to have little jumping points. And I added a **dense layer with 100 units** before last layer.



This model has **best acc at epoch 76 = 0.7797**

We have a really good diagram in loss learning curve, **no high jumping points** and **no overfitting**.

But the acc diagram shows that with more epochs **we will have overfitting**, and **jumping points still exists**.

Conclusion:

The best acc and best learning curve happened in this model so

I choose this model as our best model and I will continue testing other thing with this model.

Testing model on Test data result:

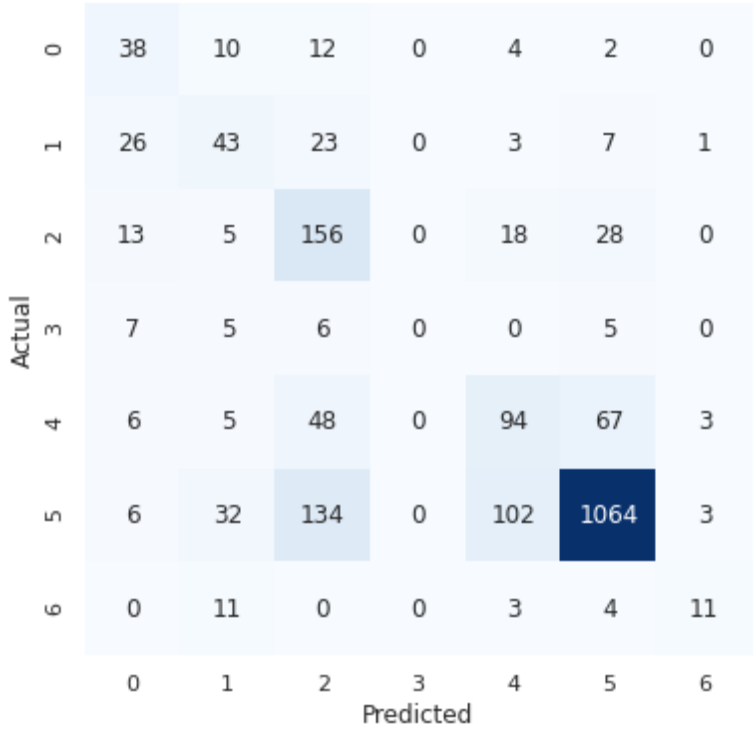
Test loss: 0.853336751461029

Test acc: 0.7012468576431274

The result of other metrics for each class:

	precision	recall	f1-score	support
0	0.40	0.58	0.47	66
1	0.39	0.42	0.40	103
2	0.41	0.71	0.52	220
3	0.00	0.00	0.00	23
4	0.42	0.42	0.42	223
5	0.90	0.79	0.85	1341
6	0.61	0.38	0.47	29
accuracy			0.70	2005
macro avg	0.45	0.47	0.45	2005
weighted avg	0.74	0.70	0.71	2005

The confusion matrix:



Conclusion:

Our **data is unbalanced** so, accuracy isn't a suitable Metrix to define the model.

Nd other results shows that our model **works good just on class 5**.

So, we need unbalancing our data first.

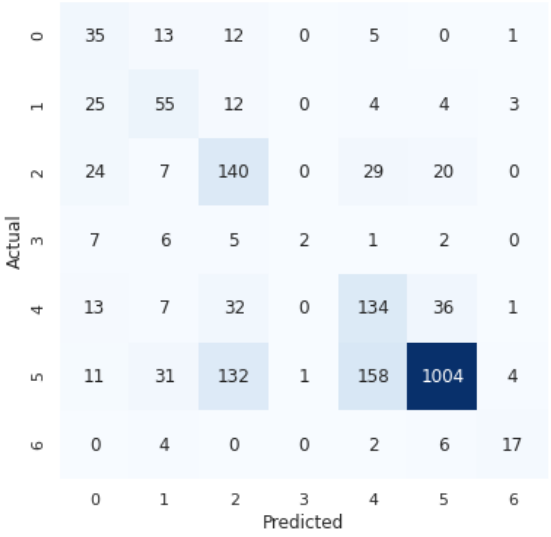
Handling unbalancing data:

Setting class weight:

With these weights:

{0: 2, 1: 2 , 3 : 3 ,4 : 2 ,5 : 1 ,6 : 2}

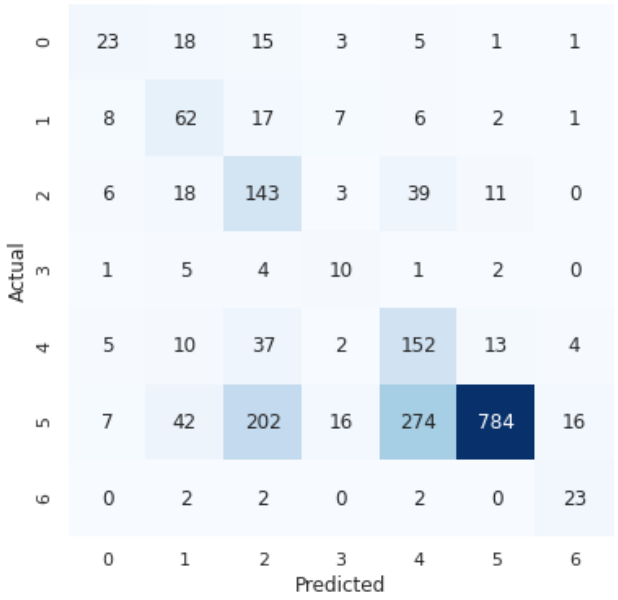
	precision	recall	f1-score	support
0	0.30	0.53	0.39	66
1	0.45	0.53	0.49	103
2	0.42	0.64	0.51	220
3	0.67	0.09	0.15	23
4	0.40	0.60	0.48	223
5	0.94	0.75	0.83	1341
6	0.65	0.59	0.62	29
accuracy			0.69	2005
macro avg	0.55	0.53	0.50	2005
weighted avg	0.77	0.69	0.71	2005



With these weights:

{0: 20,1: 20 ,2 : 20 ,3 : 50 ,4 : 20 ,5 : 1 ,6 : 50}

	precision	recall	f1-score	support
0	0.46	0.35	0.40	66
1	0.39	0.60	0.48	103
2	0.34	0.65	0.45	220
3	0.24	0.43	0.31	23
4	0.32	0.68	0.43	223
5	0.96	0.58	0.73	1341
6	0.51	0.79	0.62	29
accuracy			0.60	2005
macro avg	0.46	0.58	0.49	2005
weighted avg	0.76	0.60	0.63	2005

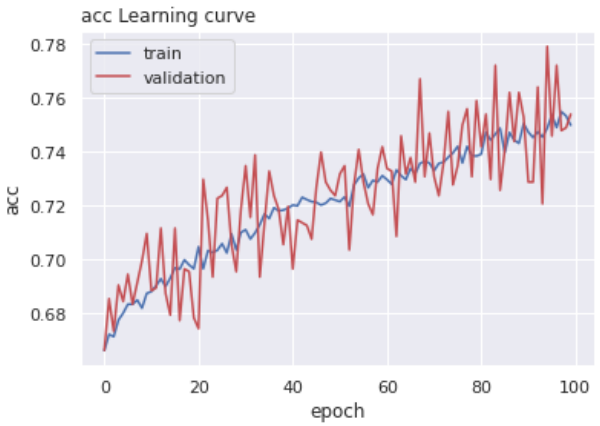


Using data generator:

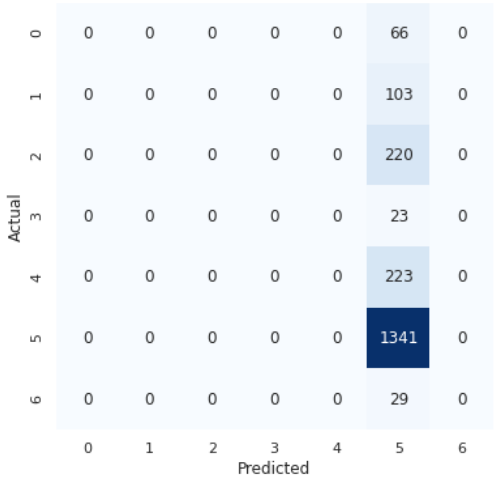
I used a simple model without any regularization nor drop out layer.

```
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', padding='same', input shape=(28, 28, 3)  ))
model.add(Conv2D(32, (3, 3), activation='relu', padding='same' ))
model.add(MaxPool2D((3, 3), strides=(2, 2)))

model.add(Conv2D(64, (3, 3), activation='relu', padding='same' ))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same' ))
model.add(MaxPool2D((2, 2), strides=(2, 2)))

model.add(Conv2D(128, (3, 3), activation='relu', padding='same' ))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(MaxPool2D((2 , 2), strides=(2, 2)))
model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
```

```
model.add(Conv2D(512, (3, 3), activation='relu', padding='same' ))
model.add(Conv2D(512, (3, 3), activation='relu', padding='same' ))
model.add(MaxPool2D((2, 2), strides=(2, 2)))

model.add(Flatten())
model.add(Dense(100, activation='relu'))
model.add(Dense(7, activation='softmax'))
```



	precision	recall	f1-score	support
0	0.00	0.00	0.00	66
1	0.00	0.00	0.00	103
2	0.00	0.00	0.00	220
3	0.00	0.00	0.00	23
4	0.00	0.00	0.00	223
5	0.67	1.00	0.80	1341
6	0.00	0.00	0.00	29
accuracy			0.67	2005
macro avg	0.10	0.14	0.11	2005
weighted avg	0.45	0.67	0.54	2005



The **overfitting** in learning curve diagram **is fitted without any regularization**.

And with this model, we reached our best's model acc = 0.7722.

And maybe by increasing epoch, we would get a better acc.

But what is more important is that this model **needs a long time to train**.

And by looking at **precision and recall** metrics, and **confusion matrix**, we get that model **isn't working good on classes with fewer samples**.

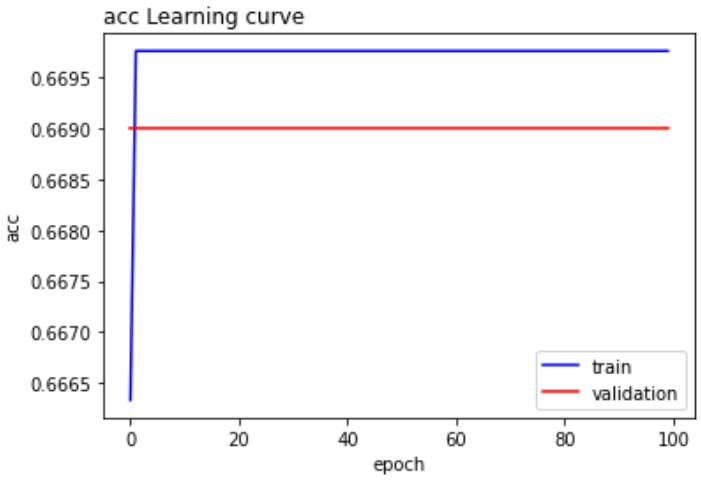
Testing other parameters

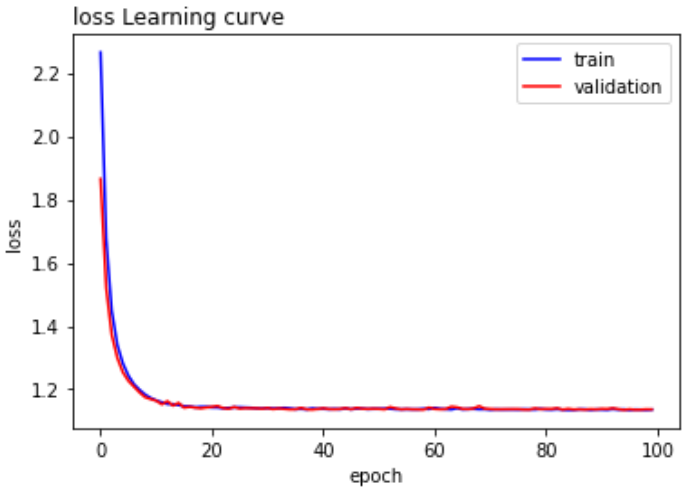
on our final model:

Activation function:

Sigmoid:

I replaced sigmoid activation function with relu in this test :



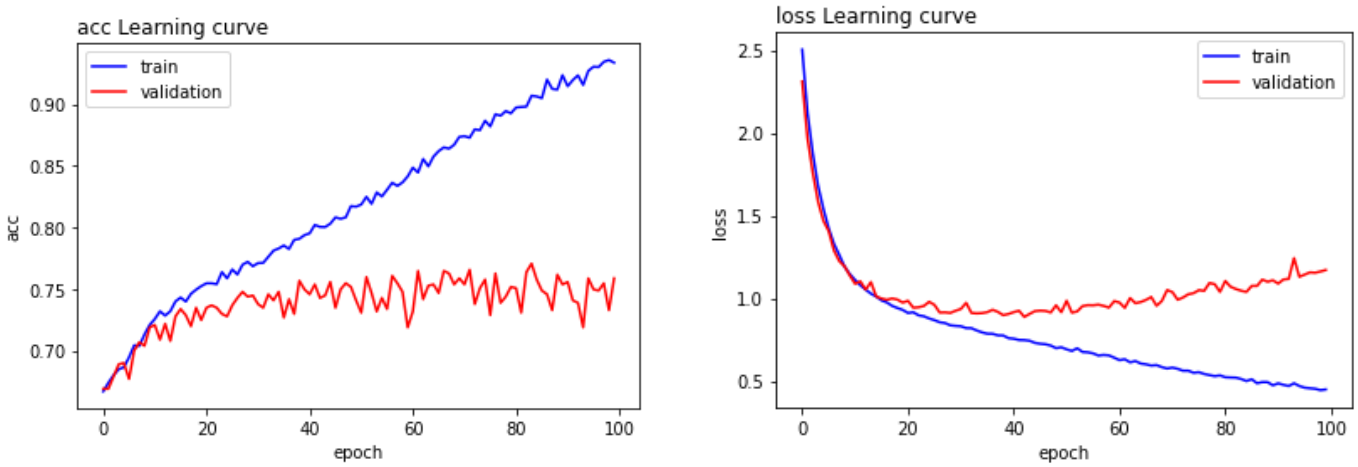


Acc learning curve is showing that, from the first epoch till last one, our score isn't improving nor decreasing, it **stopped just at acc = 0.6695**

And in loss learning curve, after some epoch, the loss **remains the same.**

Tanh:

Replacing relu activation function with tanh in CNN layers:



This model its best **<u>acc at epoch 84 = 0.7707</u>**

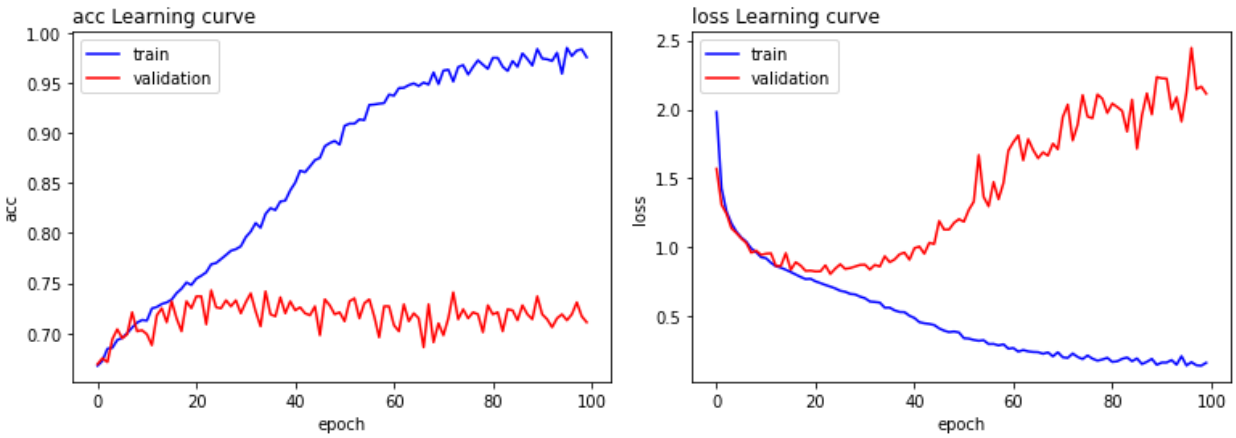
Although, the acc reached 0.77, but learning curves are showing absolute **overfitting** and we had both regularization and dropout layer in this model too.

So, comparing to **relu**, that overfitting didn't happen, **<u>relu was better option</u>**.

<u>Changing filter size in maxpoling layer:</u>

Filtee size for convoloutional layer : 5*5

Maxpolling filer size : 2*2 with stride 1 ,1

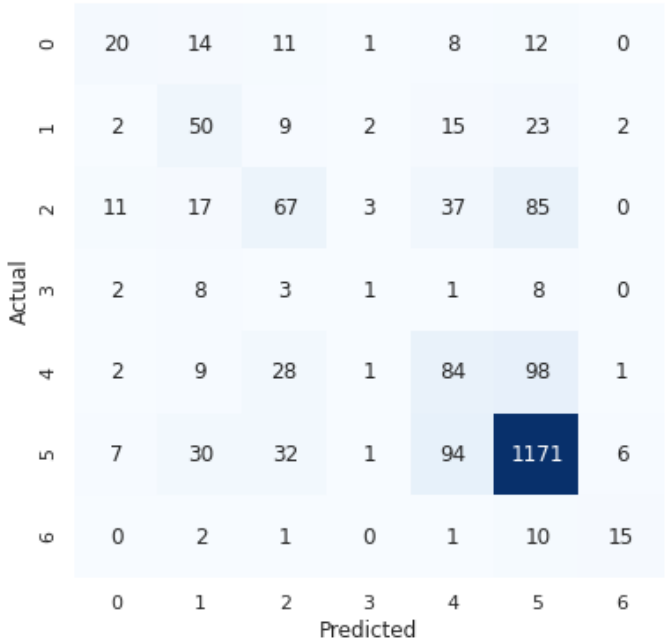


Filter size for convolutional layer**: 5*5 -** Maxpoling filer size: **2*2** with **stride 1 ,1**

Changing filter size, didn't make good result, learning curve shows **overfitting**

Test loss: 2.3560919761657715

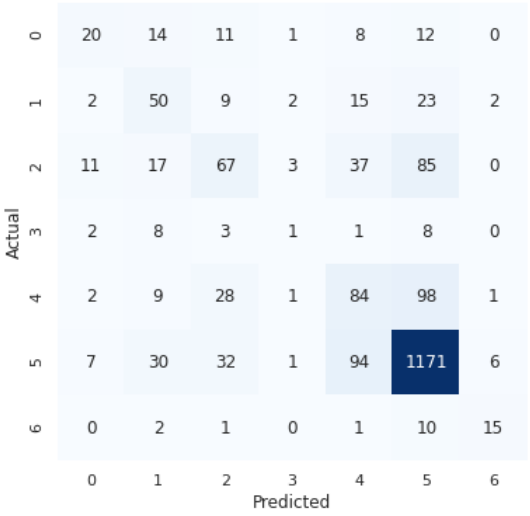
Test acc: 0.7022444009780884



	precision	recall	f1-score	support
0	0.45	0.30	0.36	66
1	0.38	0.49	0.43	103
2	0.44	0.30	0.36	220
3	0.11	0.04	0.06	23
4	0.35	0.38	0.36	223
5	0.83	0.87	0.85	1341
6	0.62	0.52	0.57	29
accuracy			0.70	2005
macro avg	0.46	0.41	0.43	2005

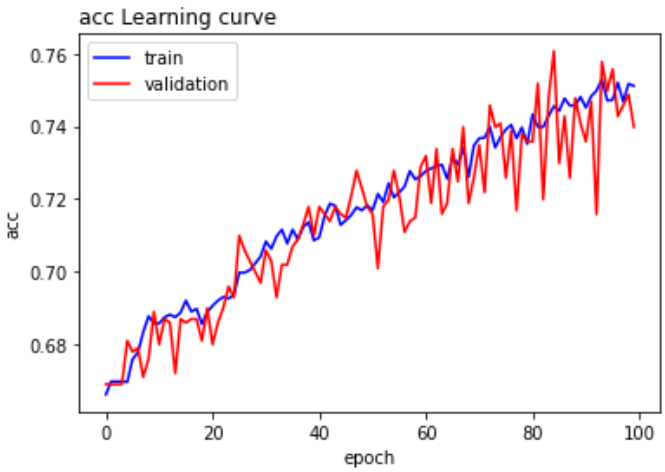
but results for recall and precision are better than our main model (the model with dropout and regularization)

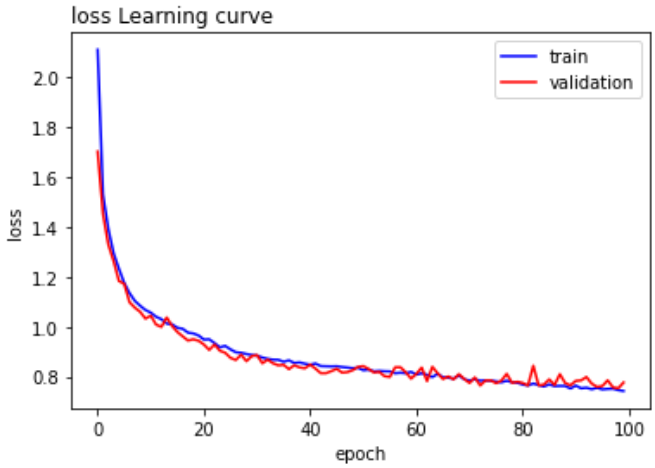
Changing filter size , didn't make

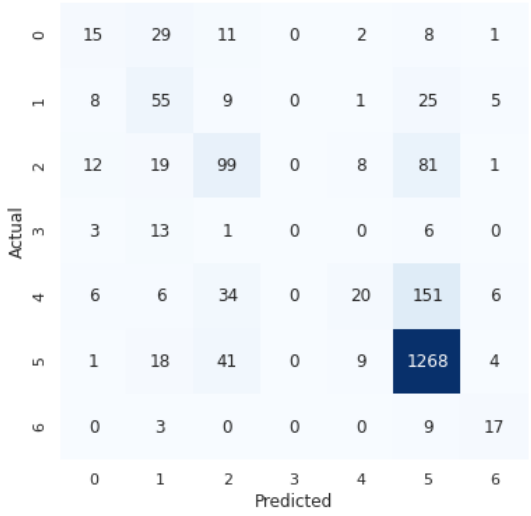


Our confusion matrix isn't good at all. we had a better confusion matrix with setting Weights .

Testing AvgPool2D:


acc Learning curve
train
validation
acc
epoch
0.76
0.74
0.72
0.70
0.68
0
20
40
60
80
100


loss Learning curve
train
validation
loss
epoch
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0
20
40
60
80
100


Actual
Predicted
0 15 29 11 0 2 8 1
1 8 55 9 0 1 25 5
2 12 19 99 0 8 81 1
3 3 13 1 0 0 6 0
4 6 6 34 0 20 151 6
5 1 18 41 0 9 1268 4
6 0 3 0 0 0 9 17
0 1 2 3 4 5 6

	precision	recall	f1-score	support
0	0.33	0.23	0.27	66
1	0.38	0.53	0.45	103
2	0.51	0.45	0.48	220
3	0.00	0.00	0.00	23
4	0.50	0.09	0.15	223
5	0.82	0.95	0.88	1341
6	0.50	0.59	0.54	29
accuracy			0.74	2005
macro avg	0.43	0.40	0.39	2005
weighted avg	0.70	0.74	0.70	2005

Test loss: 0.7824896574020386

Test acc: 0.735162079334259

Jumping points are a lot in acc learning curve and we have no better precision and recall, so Maxpoling was a better choice.