مقدمه :

در این تکلیف ما قصد داشتیم با استفاده از مدل شبکه پرسپترون چند لایه روی دیتاست CIFAR و با tune کردن پارامتر های مدل های ساخته شده، به بهترین نتیجه و دقت برسیم. این پارامتر ها یا در واقع هایپرپارامتر ها شامل مواردی نظیر تعداد لایه ها و تعداد نرون های موجود در هر لایه، optimizer استفاده شده در الگوریتم و توابع activation می باشند که باید بهترین بین آن ها انتخاب شود.

درک دیتاست CIFAR:

دیتاست موجود از لینک https://www.cs.toronto.edu/~kriz/cifar.html دانلود شده و بعد از دانلود و extract کردن به فایل cifar-10-batches-py می رسیم که ۶ فایل بایتی می باشند که ۵ فایل data_batch_X شامل بچ های داده های train می باشند و test_batch نیز شامل دادگان تست ما می باشند.

در واقع در ابتدا داده های train و test برای ما از قبل split شده اند و نیازی نیست که ما این فرآیند را انجام دهیم.

در ادامه با توجه به همین لینک و به کمک تابع unpickle آبجکت هایی که در این فایل ها که به صورت یک دیکشنری کلی ذخیره شده اند را در کد پایتون load می کنیم. آبجکت لود شده حاصله شامل چند key می باشد که ما از key به اسم data استفاده میکنیم برای ذخیره کردن ماتریس فیچر ها (X) و از key به اسم labels برای استخراج وکتور label ها (y) استفاده می کنیم. این موارد در **شکل ۱** آورده شده اند.

```
In [3]: data_batch1 = unpickle(file)
        data_batch1.keys()   # printing every batch file dict information.
Out[3]: dict_keys(['batch_label', 'labels', 'data', 'filenames'])

In [4]: # printing datas:
        print('data values for batch 1:\n')
        trainBatch1_X = data_batch1['data']    # numpy array.
        print(trainBatch1_X)
        print('\nlabels for batch 1:\n')
        trainBatch1_y = np.array(data_batch1['labels'])   # class-label vector.
        print(trainBatch1_y)

        data values for batch 1:

        [[ 59  43  50 ... 140  84  72]
         [154 126 105 ... 139 142 144]
         [255 253 253 ...  83  83  84]
         ...
         [ 71  60  74 ...  68  69  68]
         [250 254 211 ... 215 255 254]
         [ 62  61  60 ... 130 130 131]]

        labels for batch 1:

        [6 9 9 ... 1 1 5]

In [5]: print('shape of batch 1:')
        trainBatch1_X.shape     # (10000, 3072) => 10000 sampels each ( ((32 * 32)) * 3 ) (3 is for RGB channels)

        shape of batch 1:

Out[5]: (10000, 3072)
```

شکل ۱

حال در ادامه به تفصیل دیتاست میپردازیم.

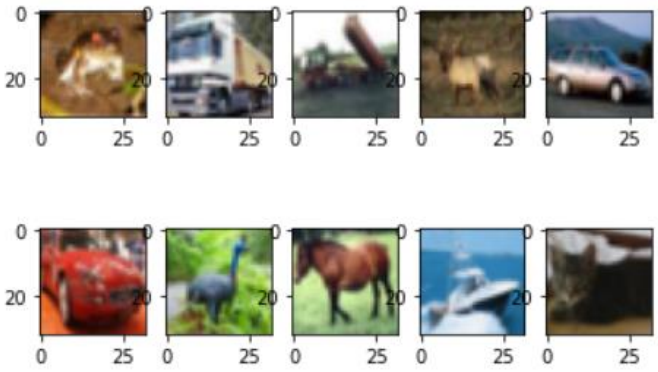
این دیتاست به طور کلی دارای ۵۰۰۰۰ سمپل برای داده های train و ۱۰۰۰۰ سمپل برای test به صورت بچ دارد. هر سمپل یک عکس رنگی است که ابعاد آن 32 * 32 می باشد.

در ابتدا اگر ابعاد ماتریس فیچر ها را برای یک بچ چاپ کنیم، حاصل 10000 * 3072 می باشد.

در واقع سمپل ها در این دیتاست به صورت جمع شده می باشند و ابعاد هر کدام در واقع یک تنسور ۳ بعدی می باشد که ابعاد آن به صورت 32*32*3 می باشد. که عدد 3 به دلیل چنل های رنگی عکس می باشد.

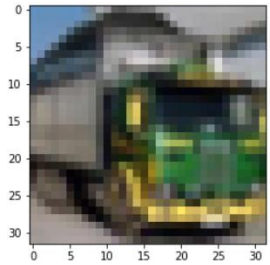
توضیحات این موارد به صورت یک مثال در عکس و توضیحات در notebook موجود می باشد.

حال برای اطمینان بیشتر و درک بیشتر از دیتا چندتا از سمپل ها را نمایش می دهیم. در ابتدا دیتاست را به یک تنسور ۴ بعدی که بعد اول همان تعداد سمپل ها می باشد و ۳ بعد بعدی آن به ترتیب عرض و طول و تعداد چنل می باشد که به صورت ثابت 32*32*3 می باشد. حال به کمک تابع imshow چند سمپل اول از بچ دیتا اول را نمایش می دهیم(**شکل ۲**).



شکل ۲

سمپل ۱۵ ام بچ اول نیز **شکل ۳** در نمایش داده شده است.



شکل ۳

در ادامه notebook نیز می آییم و در تمامی بچ ها نشان می دهیم که به ازای هر کلاس، چند رکورد داریم به طور کلی برای ۱۰ کلاس، ۵۰۰۰ سمپل داریم. نمونه آن در شکل ٤ نمایش داده شده است.

```
Label count in all train-set:

Label counts of airplane(0): 5000
Label counts of automobile(1): 5000
Label counts of bird(2): 5000
Label counts of cat(3): 5000
Label counts of deer(4): 5000
Label counts of dog(5): 5000
Label counts of frog(6): 5000
Label counts of horse(7): 5000
Label counts of ship(8): 5000
Label counts of truck(9): 5000

Label count in every batch:

batch number 1:
Label counts of airplane(0): 1005
Label counts of automobile(1): 974
Label counts of bird(2): 1032
Label counts of cat(3): 1016
Label counts of deer(4): 999
Label counts of dog(5): 937
Label counts of frog(6): 1030
Label counts of horse(7): 1001
Label counts of ship(8): 1025
Label counts of truck(9): 981
```

شکل ۴

پیش پردازش داده ها:

ما در ادامه بچ اول داده های train را به عنوان تست کردن و tune کردن مدل انتخاب میکنیم و برای پیش پردازش تعیین میکنیم.

برای پیش پردازش داده ها در ابتدا بررسی شد که missing data نداشته باشیم.

در ادامه داده های تصاویر را نرمال سازی میکنیم با توجه به اینکه بزرگترین و مقدار در پیکسل های تصاویر عدد ۲۵۵ می باشد، و با توجه به روش min-max برای نرمال کردن داده ها فقط کافی است که مقادیر ماتریس فیچر ها را تقسیم بر ۲۵۵ کنیم.

بعد از نرمالایز کردن داده ها چون مسئله ما یک مسئله چند کلاسه می باشد، و به دلیل اینکه ما می خواهیم از categorical cross entropy به عنوان تابع loss استفاده کنیم، باید داده های وکتور لیبل ها را one hot encode کنیم تا یک نمایش باینری به صورت کد شده برای هر کلاس داشته باشیم.

این پیش پردازش را به کمک تابع to_catagorical داخل keras انجام میدهیم. خروجی یک ماتریس به ابعاد 10* 10000 خواهد بود در واقع به ازای هر ورودی یا سمپل دیتا ما یک نمایش باینری با ۱۰ بیت برای هر لیبل دیتا در نظر گرفته ایم.

و در ادامه برای پخش کردن توزیع داده ها از unison_suffle استفاده می کنیم و داده ها را به صورت رندوم suffle می کنیم. این موارد در **شکل ۵** و **شکل ٦** قابل رویت می باشند.

Data Preprocessing:

In [95]:
```
from tensorflow.keras.utils import to_categorical

def normalizeData(data):
    # min_val = np.min(data) => 0   (pixel with value zero in its channel.)
    # max_val = np.max(data) => 255   (pixel with value 255 in its channel.)
    # data = (data-min_val) / (max_val-min_val)
    return data / 255.

def oneHotEncode(data):
    return to_categorical(data)

def unison_shuffle(a, b):
    index = np.random.permutation(a.shape[0])
    return a[index], b[index]
```

In [96]:
```
# trainBatch1_X = re_ScaleData(trainBatch1_X)

trainBatch1_X, trainBatch1_y = unison_shuffle(trainBatch1_X, trainBatch1_y)
trainBatch1_X = normalizeData(trainBatch1_X)    # normalize the train data.
trainBatch1_y = oneHotEncode(trainBatch1_y)    # one-hot encoding

# printing 5 samples(each 32 * 32 * 3) from train.
print('normalized dataBatch1:\n', trainBatch1_X[:5])
print('\none hot encoded lables:\n', trainBatch1_y[:5])
```

شکل ۵

```
normalized dataBatch1:
 [[0.49019608 0.48627451 0.47058824 ... 0.61176471 0.6745098  0.62745098]
 [0.96862745 0.96078431 0.96078431 ... 0.03529412 0.04313725 0.04313725]
 [0.07058824 0.05882353 0.01960784 ... 0.12156863 0.20784314 0.09411765]
 [0.42352941 0.4        0.51372549 ... 0.36078431 0.35686275 0.43137255]
 [0.27058824 0.22352941 0.16078431 ... 0.48235294 0.5254902  0.49803922]]

one hot encoded lables:
 [[0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]]
```

شکل ۶

در ادامه نیز پیش پردازش train-validation را انجام میدهیم، این کار را تابع train_validation_split برای ما انجام میدهد. کاری که می کند این است با گرفتن X و y و درصد میزان داده انتخابی برای داده آموزش را گرفته و داده را به بخش های آموزش و ارزیابی برای تعیین پارامتر های مدل تقسیم می کنیم(**شکل ۷**)

Creating train-set and validation-set from dataBatch1:

```
In [97]: def train_validation_split(X, y, rate):

             threshold = int(trainBatch1_X.shape[0] * rate)

             train_X = X[: threshold]
             validation_X = X[threshold :]
             train_y = y[ : threshold]
             validation_y = y[threshold : ]

             return train_X, train_y, validation_X, validation_y
```

```
In [98]: # we have large data, so we select 15 percent of data to be the validation-set.
         train_X, train_y, validation_X, validation_y = train_validation_split(trainBatch1_X, trainBatch1_y, 0.85)

         print("\nsplited validation_X data: \n", validation_X[ : 5])
         print("\nsplited validation_y data:\n ", validation_y[ : 5])

         print("\nsplited train_X data: \n", train_X[ : 5])
         print("\nsplited train_y data:\n ", train_y[ : 5])
```

شکل ۷

در این جا چون داده های زیادی داریم تعداد داده های train را حدود ۸۵ درصد و تعداد داده های validation را حدود ۱۵ درصد در نظر گرفته ایم.

در ادامه به سراغ tune کردن و تست کردن مدل های مختلف می رویم.

مدل ها:

در ابتدا چند مدل ساده تک لایه را با activation function های متفاوت با بهینه ساز SGD امتحان میکنیم.

این مورد در **شکل ۸** و **شکل ۹** مشخص می باشند.

```
INPUT_DIM = 3072   # featurs of our problem.

model_lst = []
#---------------------- single layer models with diffrent activation functions.
neural_model1 = create_neural_Model(1, (10, ), ('relu',), INPUT_DIM)
model_lst.append(neural_model1)

neural_model2 = create_neural_Model(1, (10, ), ('sigmoid',), INPUT_DIM)
model_lst.append(neural_model2)

neural_model3 = create_neural_Model(1, (10, ), ('tanh',), INPUT_DIM)
model_lst.append(neural_model3)

neural_model4 = create_neural_Model(1, (10, ), ('softplus',), INPUT_DIM)
model_lst.append(neural_model4)

neural_model5 = create_neural_Model(1, (10, ), ('softsign',), INPUT_DIM)
model_lst.append(neural_model5)

neural_model6 = create_neural_Model(1, (10, ), ('selu',), INPUT_DIM)
model_lst.append(neural_model6)

neural_model7 = create_neural_Model(1, (10, ), ('elu',), INPUT_DIM)
model_lst.append(neural_model7)

neural_model8 = create_neural_Model(1, (10, ), ('softmax',), INPUT_DIM)
model_lst.append(neural_model8)

neural_model9 = create_neural_Model(1, (10, ), ('linear',), INPUT_DIM)
model_lst.append(neural_model9)
```

شکل ۸

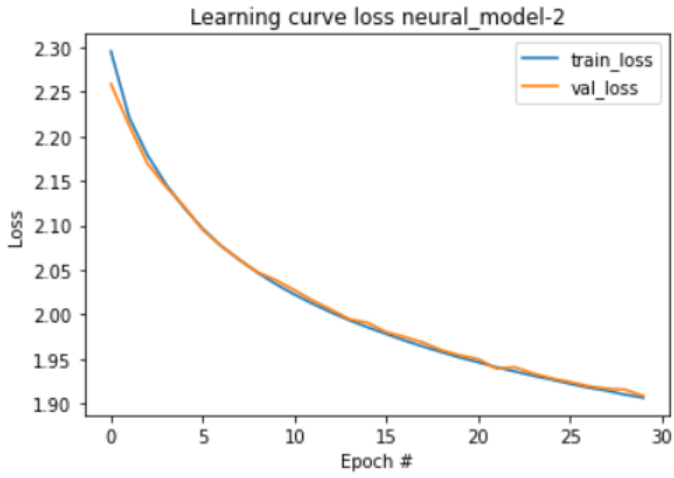
```
In [103]: # neural_model8.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_crossentropy', metrics=['accuracy'])

opt = SGD(learning_rate=0.0001)
compileModels(model_lst, opt)
```

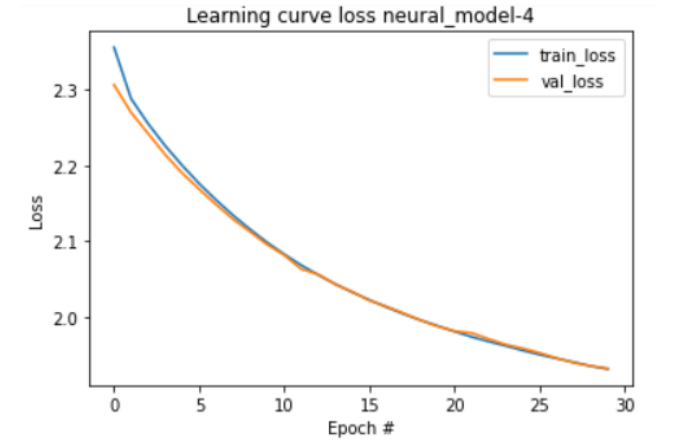
```
In [104]: NUM_EPOCHS = 30
BATCH_SIZE = 10

history_lst = []
hs_lst = trainModels(model_lst, NUM_EPOCHS, BATCH_SIZE, train_X, train_y, (validation_X, validation_y))
# hs = neural_model8.fit(train_X, train_y, batch_size = BATCH_SIZE, epochs = NUM_EPOCHS, validation_data = (validation_X, validat
```

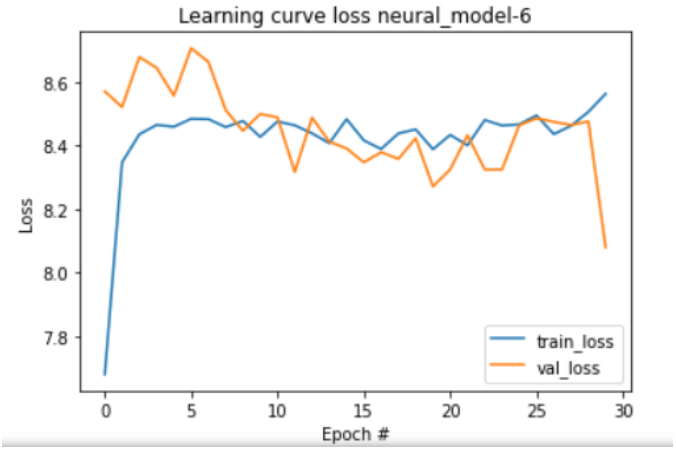
```
Model 1 started training:

Epoch 1/30
850/850 [==============================] - 1s 1ms/step - loss: 2.8215 - accuracy: 0.0976 - val_loss: 2.3764 - val_accuracy:
0.1013
Epoch 2/30
850/850 [==============================] - 1s 1ms/step - loss: 2.3463 - accuracy: 0.0976 - val_loss: 2.3172 - val_accuracy:
0.1033
Epoch 3/30
850/850 [==============================] - 1s 1ms/step - loss: 2.3022 - accuracy: 0.0989 - val_loss: 2.2880 - val_accuracy:
0.1053
Epoch 4/30
850/850 [==============================] - 1s 1ms/step - loss: 2.2774 - accuracy: 0.1074 - val_loss: 2.2703 - val_accuracy:
0.1180
Epoch 5/30
850/850 [==============================] - 1s 1ms/step - loss: 2.2612 - accuracy: 0.1333 - val_loss: 2.2584 - val_accuracy:
0.1520
Epoch 6/30
```

شکل ۹

در ادامه برای این مدل تستی بسیار سبک learning curve را رسم میکنیم تا در ادامه بین این ۹ مدل چندتا رو بازم به عنوان بهترین انتخاب کنیم و ادامه دهیم.

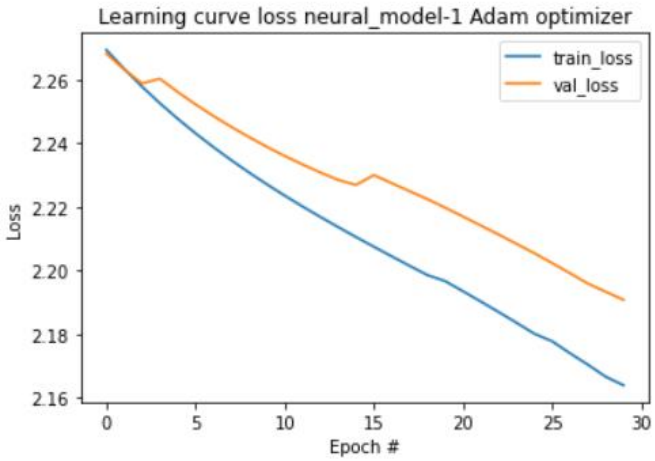


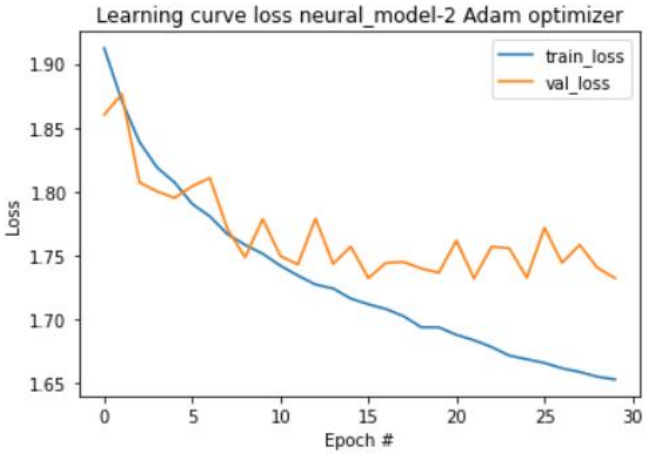


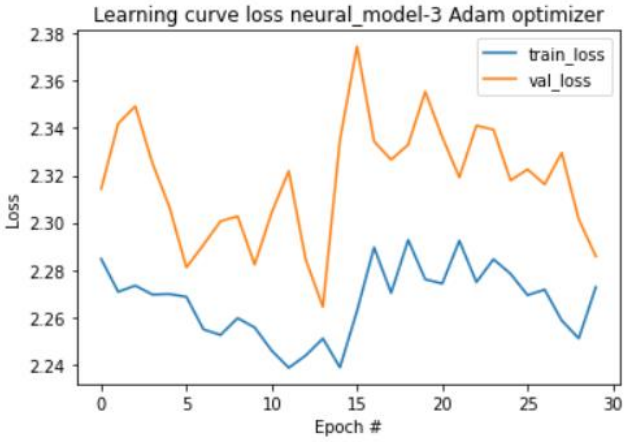


نمودار های بیشتر در notebook برای ۹ مدل به ترتیب هم برای loss و هم برای accuracy نیز رسم شده اند و در آن موجود می باشند. همانطور که در نمودار ها مشخص می باشد، مدل تک لایه شماره ۲ با تابع سیگموید و مدل شماره ۴ با تابع softplus و مدل شماره ۸ با تابع softmax بهترین عملکرد را نسبت به مدل های دیگر دارند به عنوان مثال در مدل شماره ۶ خطای مدل در بخش داده های آموزشی به مرور زمان حتی افزایش یافته است و این به معنای این است که مدل تا حدی underfit شده و روی داده آموزش هم نتوانسته به خوبی آموزش پیدا کند.

در ادامه نیز بین این ۳ مدل بار دیگر با بهینه ساز Adam تست میکنیم در ادامه با نتایج جالبی روبرو میشویم با توجه به نمودار ها مدل های ۱ و ۲ که در مرحله قبل مدل های ۲ و ۴ بودن، عملکرد بهتری نسبت به مدل سوم که مدل شماره ۸ بود داشته است، دلیل آن این است که مدل های train شده خروجی هایشان بیانگر احتمال عضویت در یک کلاس مشخص نمی باشد. در صورتی که در تابع softmax خروجی بیانگر احتمال عضویت در یک کلاس است به همین دلیل حتی با توجه به اینکه مدل تا حد قابل توجهی underfit شده است، از آن فقط و فقط در لایه آخر هر شبکه عصبی استفاده خواهیم کرد


Learning curve loss neural_model-1 Adam optimizer
train_loss
val_loss
Loss
Epoch #


Learning curve loss neural_model-2 Adam optimizer
train_loss
val_loss
Loss
Epoch #


Learning curve loss neural_model-3 Adam optimizer
train_loss
val_loss
Loss
Epoch #

در ادامه به دنبال tune کردن مدل و مشخص کردن پارامتر های نهایی برای بیش از ۱ لایه می پردازیم.

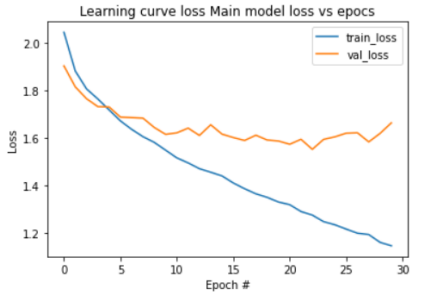
در ابتدا چون می خواهیم بین چند مدلی که بهترین کارکرد را تا این جای تست کردن مدل ها داشتن را اطلاعات پارامتر هایشان را نگه داریم به ازای هایپر پارامتر ها و ویژگی هایشان یک شی جدید می سازیم و اطلاعات پارامتر ها را در آن شی ذخیره می کنیم. برای ساختن این اشیاء برای ذخیره سازی اطلاعات از کلاس Model_Info استفاده می کنیم که یکسری ویژگی ها برای مدل های امتحانی را در خود ذخیره می کنند(**شکل ۱۰**).

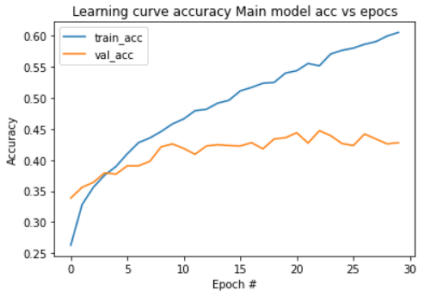
model parameters info class:

```
In [100]: # this class stores some information for using in some model evaluation.
          class Model_Info:
                  def __init__(self, layers, neurons, activation_funcs, dim, opt, loss_func, metric, epochs, batch_size):
                      self.layers = layers
                      self.neurons = neurons
                      self.activation_funcs = activation_funcs
                      self.dim = dim
                      self.opt = opt
                      self.loss_func = loss_func
                      self.metric = metric
                      self.epochs = epochs
                      self.batch_size = batch_size
```

شکل ۱۰

در ادامه نیز یکی از مدل های چند لایه به عنوان تست آورده شده است که برای آن هم confusion ماتریس و هم learning curve و هم به ازای هر کلاس دقت و recall و precision محاسبه شده است این مدل یک مدل تخمینی با پارامتر های نسبتا بهینه است که اطلاعات آن را نیز برای ارزیابی نهایی در روش k-fold cross validation برای انتخاب مدل نهایی ذخیره می کنیم. این مدل نسبتا سریع train می شود و دقت train و validation خوبی دارد. این مدل حاصل تست چند مدل و تعیین بعضی از هایپر پارامتر ها می باشد.





مشخص است که مدل با توجه به نمودار acc و loss آن نسبت به تعداد epoch ها تا حدی بعد از حدود ۱۵ epoch به حالت overfit رسیده است به همین دلیل می توانیم تا حدی برای جلو گیری از overfit تعداد epoch ها را کاهش بدهیم.

```
cm = create_confusionMatrix(y_pred, validation_y)
print('confusion matrix for first data batch validation:\n')
cm
```

```
confusion matrix for first data batch validation:


array([[ 73.,  12.,   4.,   2.,   1.,   0.,   0.,   2.,  35.,   4.],
       [  5., 107.,   2.,   0.,   2.,   5.,   2.,   3.,  15.,  18.],
       [ 28.,   9.,  45.,   6.,  17.,  16.,   3.,   9.,  17.,   4.],
       [ 14.,  16.,  10.,  35.,   7.,  46.,   8.,  12.,  17.,  15.],
       [  9.,   5.,  31.,   3.,  41.,  11.,   3.,  19.,   7.,   5.],
       [  5.,   8.,   9.,  14.,   7.,  47.,   6.,   7.,  12.,   5.],
       [  8.,  10.,  17.,  17.,  20.,  21.,  47.,   5.,  12.,   5.],
       [ 16.,   9.,   2.,   3.,   9.,  14.,   1.,  89.,   9.,  16.],
       [ 19.,  12.,   0.,   2.,   1.,   0.,   1.,   2.,  94.,   7.],
       [ 11.,  45.,   3.,   2.,   1.,   2.,   0.,   5.,  19.,  64.]])
```

در confusion ماتریس بالا حالت های ردیفی (عمودی) لیبل های اصلی و حالت های افقی لیبل های predict شده توسط مدل می باشد. توجه شود که هر دارایه در قطر اصلی ماتریس میزان صحت predict می باشد که با لیبل متناظر آن در دیتاست مطابقت دارد.

```
acc, per, recall = calculate_metrics(y_pred, validation_y, validation_X)
for i in range(validation_y.shape[1]):
    print('on label {}, acc={:0.2f}, precision={:0.2f}, recall={:0.2f}'.format(i, acc[i], per[i], recall[i]))
```

```
on label 0, acc=88.33, precision=38.83, recall=54.89
on label 1, acc=88.13, precision=45.92, recall=67.30
on label 2, acc=87.53, precision=36.59, recall=29.22
on label 3, acc=87.07, precision=41.67, recall=19.44
on label 4, acc=89.47, precision=38.68, recall=30.60
on label 5, acc=87.47, precision=29.01, recall=39.17
on label 6, acc=90.73, precision=66.20, recall=29.01
on label 7, acc=90.47, precision=58.17, recall=52.98
on label 8, acc=87.53, precision=39.66, recall=68.12
on label 9, acc=88.87, precision=44.76, recall=42.11
```

در شکل بالا نیز متریک های accuracy و precision و recall به ازای هر کلاس در بچ شماره ۱ داده های train نمایش داده شده اند.

K-fold cross validation:

از این روش برای ارزیابی نهایی و مقایسه چند مدل بین مدل های پیشین که کارایی نسبتا بهتری نسبت به دیگری داشته اند استفاده شده است. از این روش نیز می توانستیم در ابتدا استفاده کنیم ولی از آن جایی که دیتاست نسبتا حجیمی داشتیم، و می توانستیم به راحتی به صورت ایستا یا همان روش train-test-split این کار را انجام بدهیم و داده validation قابل ارزیابی کردن میشد و حتی روش خیلی سریع تری می باشد، K-fold cross validation استفاده نشد. دقت شود تا به این جا همه چیز رو دیتا بچ اول فایل های train انجام گرفته است. برای ارزیابی مدل های نهایی و انتخاب بهترین آن ها با توجه به میانگین accuracy آن ها در آخرین مرحله یا epoch هر مدل، مدل نهایی با بهترین هایپر پارامتر ها را پیدا می کنیم.

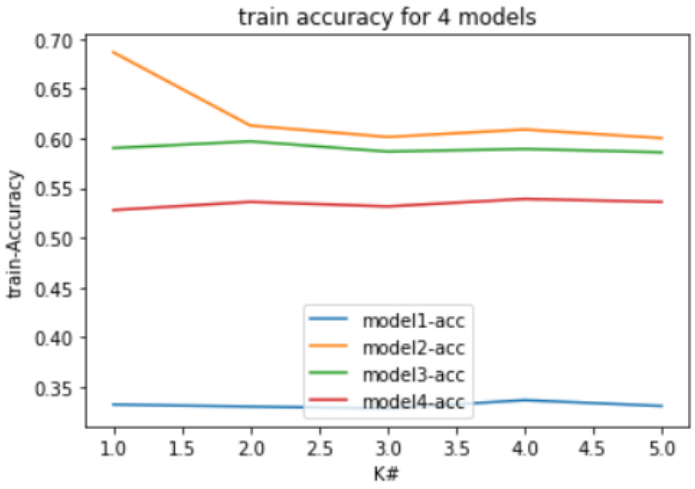
در بخش k-fold مقدار k در نظر گرفته شده برای fold ها ۵ می باشد و ۴ مدل که اطلاعات آن ها از شی ایی از جنس Model_info بود ذخیره کرده بودیم را هر دفعه و برای fold ها train میکنیم و اطلاعات accuracy را هم برای داده های train و هم برای داده های validation انجام می دهیم.

در این فرآیند نیز برای هر کلاس داده های validation در هر k و برای هر مدل متریک های recall و precision و accuracy هر کلاس نیز محاسبه می شود و در نهایت بعد از انجام k-fold، از آن ها میانگین میگیریم(**شکل ۱۱**) .

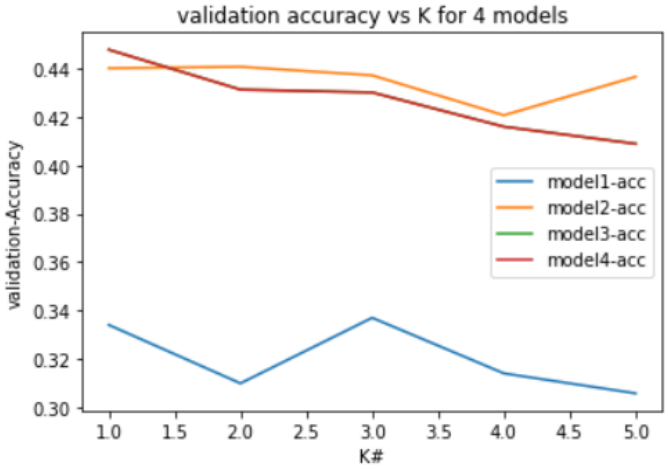
```
In [126]: avg_dict = {'model1': 0, 'model2': 0,
                      'model3': 0, 'model4': 0}

          # ------------------------ calculating accuracy for every model.
          for key in acc_scores.keys():
              avg_dict[key] = np.mean(acc_scores[key])

          # printing the best model according to their k-fold cross validations mean.
          avg_dict = {k: v for k, v in sorted(avg_dict.items(), key=lambda item: item[1])}
          avg_dict

Out[126]: {'model1': 86.40470588235294,
           'model3': 88.53411764705882,
           'model4': 88.5435294117647,
           'model2': 88.69882352941177}

In [127]: avg_dict = {'model1': 0, 'model2': 0,
                      'model3': 0, 'model4': 0}

          # ------------------------ calculating accuracy for every model.
          for key in recall_scores.keys():
              avg_dict[key] = np.mean(recall_scores[key])

          # printing the best model according to their k-fold cross validations mean.
          avg_dict = {k: v for k, v in sorted(avg_dict.items(), key=lambda item: item[1])}
          avg_dict

Out[127]: {'model1': 32.01403217357623,
           'model4': 42.707195311851265,
           'model3': 42.82370750368024,
           'model2': 43.590188477648184}
```

شکل ۱۱

همانطور که گفته شد accuracy برای کل مدل هم برای داده های آموزشی و هم برای داده های validation نیز در ادامه آورده شده اند.



شکل ۱۲



شکل ۱۳

شکل ۱۲و **شکل ۱۳** نمودار های توزیع accuracy برای ۴ مدل مورد ارزیابی برای داده های train و validation می باشند. محور افقی k و محور عمودی accuracy هر مدل می باشد. با نگاه به این ۲ نمودار متوجه می شویم که پارامتر های مدل شماره ۲ در k-fold بهتر عمل کرده است حتی از مدلی که قبلا به صورت حدسی(مدل شماره ۴) انتخاب کرده بودیم نیز بهتر جواب داده است. و حتی می توان دید که مدل تک لایه ایی دقت بسیار کمی هم در داده ها ی train و هم در داده های validation دارد و همواره در حالت underfit قرار دارد.

```
In [130]: # avg of models in k-fold for training set

          avg_dict = {'model1': 0, 'model2': 0,
                      'model3': 0, 'model4': 0}

          for key in model_acc_train.keys():
              avg_dict[key] = np.mean(model_acc_train[key])

          # sorting keys(model names) according to values
          avg_dict = {k: v for k, v in sorted(avg_dict.items(), key=lambda item: item[1])}
          avg_dict

Out[130]: {'model1': 0.33173528909683225,
           'model4': 0.5342058658599853,
           'model3': 0.5899117588996887,
           'model2': 0.6220588326454163}

In [131]: # avg of models in k-fold for validation set

          avg_dict = {'model1': 0, 'model2': 0,
                      'model3': 0, 'model4': 0}

          for key in model_acc_validation.keys():
              avg_dict[key] = np.mean(model_acc_validation[key])

          # sorting keys(model names) according to values
          avg_dict = {k: v for k, v in sorted(avg_dict.items(), key=lambda item: item[1])}
          avg_dict

Out[131]: {'model1': 0.32023529410362245,
           'model3': 0.42670588493347167,
           'model4': 0.42670588493347167,
           'model2': 0.43494117856025694}
```

شکل ۱۴

در **شکل ۱۴** نیز برای هر مدل از دیتا های train و validation میانگین میگیریم و مرتب میکنیم و همانطور که پیش تر از روی نمودار ها توضیح داده شد پارامتر های مدل ۲ نسبتا از مدل های دیگر بهتر عمل می کنند.

با توجه به چندین ارزیابی و نتایج نهایی با تست k-fold cross validation مدل نهایی را با پارامتر های model4 در نمودار میسازیم و کل داده های بچ ها را با آن train می کنیم و سپس با استفاده از داده های test که در فایل test_batch می باشد مدل نهایی را تست می کنیم.

مدل نهایی و ارزیابی آن:

در ابتدا ما داده های کل بچ ها پیش پردازش می کنیم و همان فرایند هایی که روی بچ اول رفتیم را نیز روی کل داده های بچ ها اعمال می کنیم. در ادامه دادگان test را در کد پایتون load می کنیم و روی آن ها نیز عملیات پیش پردازش را اعمال می کنیم.

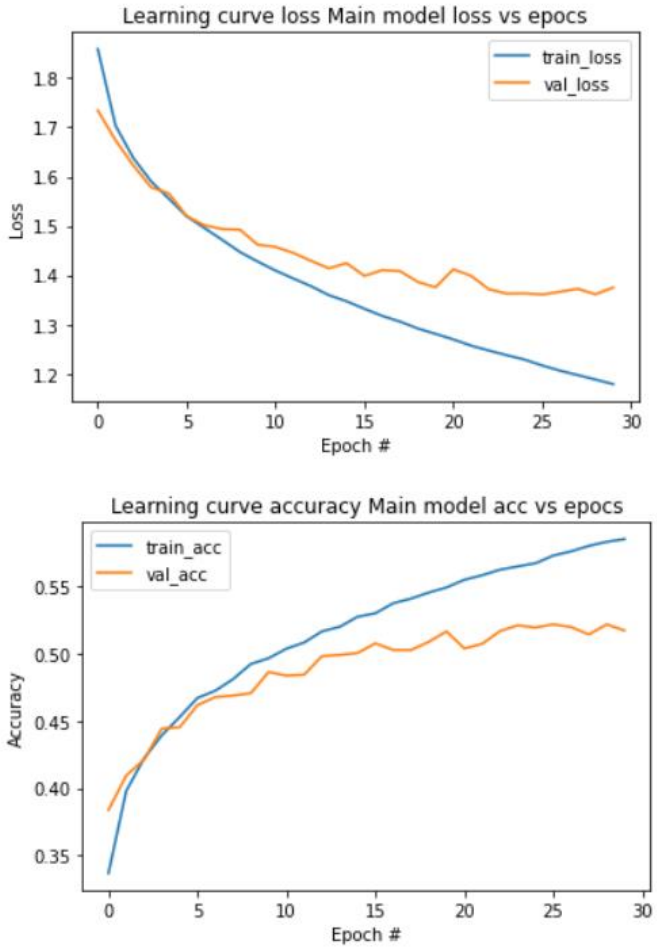
در ادامه مدلی جدید با پارامتر و هایپر پارامتر های مدل شماره ۲ که در بخش قبل صحبت شد می سازیم. و آن را با دادگان کل بچ ها train می کنیم(**شکل ۱۵**).

Train main_model with whole CIFAR dataset:

```
In [144]: main_model = create_neural_Model(info2.layers, info2.neurons, info2.activation_funcs
                                           , info2.dim,)
          main_model.compile(optimizer = info2.opt, loss = info2.loss_func, metrics = info2.metric)

          main_hs = main_model.fit(all_trainData_X, all_trainData_y, batch_size = info2.batch_size,
                                   epochs = info2.epochs, validation_data = (test_X, test_y))
```

شکل ۱۵

در ادامه پس از اتمام فرآیند آموزش مدل مجددا learning curve این مدل را رسم میکنیم(**شکل ۱۶**).



شکل ۱۶

همانطور که دیده میشود مدل تا حدودی دیتا های تست را نیز به خوبی جواب داده است ولی بعد از حدودا ۲۰ epoch مدل کم کم به سمت overfit شدن می رود.

در ادامه ارزیابی های نهایی نظیر confusion ماتریس و نمایش متریک های مختلف روی داده های تست آورده شده اند.

در **شکل ۱۷** confusion ماتریس مدل نهایی و در **شکل ۱۸** متریک ها به ازای هر کلاس آورده شده است.

Final evaluation of model:

```
y_pred = main_model.predict(test_X)

cm = create_confusionMatrix(y_pred, test_y)
print('confusion matrix for final model:\n')
cm
```

```
confusion matrix for final model:


array([[601.,  20.,  63.,  17.,  40.,  34.,  19.,  29.,  87.,  90.],
       [ 35., 543.,  17.,  18.,   9.,  22.,  20.,  25.,  60., 251.],
       [ 72.,   5., 422.,  60., 118., 123.,  91.,  70.,  12.,  27.],
       [ 22.,  11.,  79., 305.,  44., 299., 113.,  52.,  23.,  52.],
       [ 38.,   4., 129.,  61., 421.,  77., 134.,  90.,  22.,  24.],
       [ 15.,   5.,  75., 168.,  52., 512.,  62.,  62.,  14.,  35.],
       [  7.,   7.,  65.,  76.,  87.,  81., 617.,  16.,  12.,  32.],
       [ 27.,   6.,  37.,  48.,  71., 108.,  22., 614.,   9.,  58.],
       [109.,  47.,  16.,  18.,  23.,  34.,   8.,  19., 618., 108.],
       [ 28.,  92.,   7.,  24.,   9.,  35.,  12.,  34.,  39., 720.]])
```

شکل ۱۷

```
acc, per, recall = calculate_metrics(y_pred, test_y, test_X)
for i in range(test_y.shape[1]):
    print('on label {}, acc={:0.2f}, precision={:0.2f}, recall={:0.2f}'.format(i, acc[i], per[i], recall[i]))
```

```
on label 0, acc=92.48, precision=63.00, recall=60.10
on label 1, acc=93.46, precision=73.38, recall=54.30
on label 2, acc=89.34, precision=46.37, recall=42.20
on label 3, acc=88.15, precision=38.36, recall=30.50
on label 4, acc=89.68, precision=48.17, recall=42.10
on label 5, acc=86.99, precision=38.64, recall=51.20
on label 6, acc=91.36, precision=56.19, recall=61.70
on label 7, acc=92.17, precision=60.73, recall=61.40
on label 8, acc=93.40, precision=68.97, recall=61.80
on label 9, acc=90.43, precision=51.54, recall=72.00
```

شکل ۱۸

دقت نهایی مدل برای داده های train مقدار 0.6617 و برای داده های test مقدار 0.5373 می باشد.

برای دیدن بهترین دقت های بدست آمده برای این دیتاست می توان از به سایت های

https://paperswithcode.com/sota/image-classification-on-cifar-10

و https://benchmarks.ai/cifar-10

مراجعه کرد با توجه به این سایت ها بهترین دقت بدست آمده برای این دیتاست با مدل های Fractional Max-Pooling با دقت ۹۶.۵۳ درصد و مدل Big Transfer (BiT): General Visual Representation Learning حدود ۹۹.۷۳ می باشد. دلیل اینکه مدل MLP نتوانست دقت مناسبی کسب کند این است که به طور کلی MLP برای داده های تصویر و مخصوصا تصاویر رنگی مناسب نمی باشد و به خاطر غیر متمرکز بودن دیتاست

CIFAR 10 وزن ها ممکن است به درستی آپدیت نشوند و به نتیجه مطلوب نرسیم. برای بهبود می توانیم از شبکه های کانوولوشنی استفاده کنیم تا این دقت به حدودا نزدیک ۸۴ درصد برسد که این قابل قبول تر از مدل MLP برای این دیتاست می باشد.

- ✓ **توجه شود که ممکن است یکسری از نمودار ها و خروجی های اسکرین شات های قرار داده شده با notebook تفاوت داشته باشند به دلیل اینکه بعد از نوشتن این گزارش تغییرات بسیار کمی در کد ایجاد شده و رندونم بودن shuffle دیتا ها بعد از ران کردن کل notebook تاثیر ناچیزی در خروجی و نمودار ها خواهد گذاشت .**

- ✓ **تمام توضیحات و نمودار ها و کد ها داخل نوت بوک در پوشه notebook فایل ارسالی با نام MLP_CIFAR.ipynb موجود میباشد.**