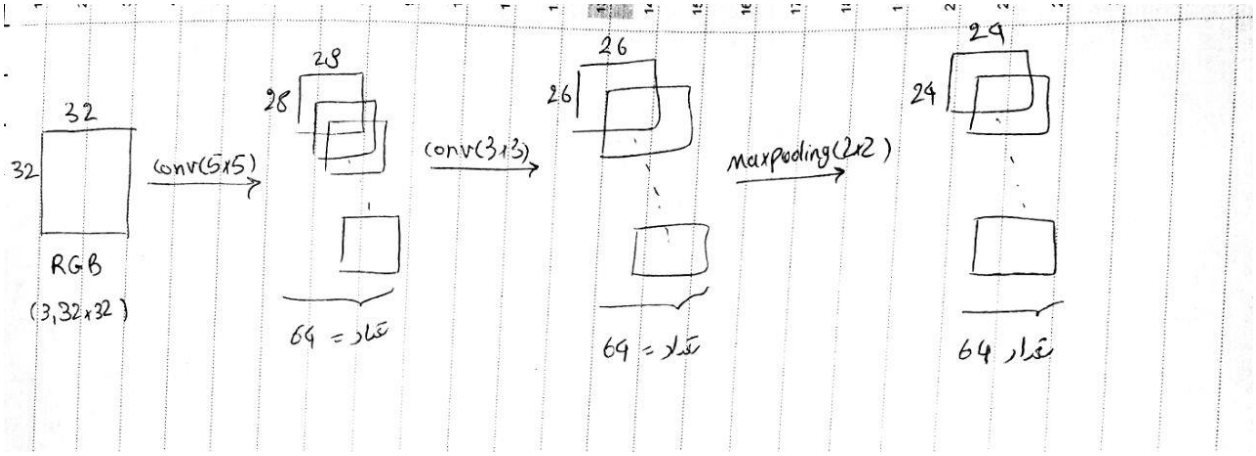
سوال ۱)

الف:



لایه اول: ۳ورودی ، ۶۴ خروجی ، $64 * (1 + (3 * 5 * 5))$پارامتر

لایه دوم: ۶۴ ورودی ، ۶۴ خروجی$64 * (1 + (3 * 3 * 3))$ پارامتر

لایه سوم: ۶۴ ورودی، ۶۴ خروجی

ب: dilation تکنیکی است که ورودی را با قرار دادن فاصله هایی بین عناصر متوالی آن گسترش می دهد. به عبارت ساده تر، همان کانولوشن است، اما شامل پرش پیکسل است، به طوری که ناحیه بزرگتری از ورودی را پوشش می دهد.

Stride : طول گام نحوه چرخش فیلتر به دور ولوم ورودی را کنترل می کند.

ج : مزایا: مکس پولینگ عملیاتی است که برای کاهش مقیاس تصویر در صورت استفاده نشدن به کار میرود و مهمترین ویژگی ها استخراج میشود.

معایب: هزینه فرایند آموزش آن زیاد است.

سوال ۲)

1. Import dependencies

```
[1]  import tensorflow
     import numpy as np
     from tensorflow.keras.layers import Input, Conv2D, Dense, Flatten, Dropout
     from tensorflow.keras.layers import GlobalMaxPooling2D, MaxPooling2D
     from tensorflow.keras.layers import BatchNormalization
     from tensorflow.keras.models import Model
     from tensorflow.keras.datasets import cifar10
     import matplotlib.pyplot as plt
     from tensorflow import keras
     from sklearn.utils import shuffle
     from keras.utils import np_utils
     from keras.models import Sequential
```

برای شروع کار کتابخانه هایی که در ادامه به آنها نیاز خواهیم داشت را فراخوانی میکنیم.

2. Load data

```
[2]  def load_data(dataset):
         cifar10 = keras.datasets.cifar10
         (x_train, y_train), (x_test, y_test) = cifar10.load_data()

         return x_train, y_train, x_test, y_test
```

```
[3]  x_train, y_train, x_test, y_test = load_data(cifar10)
     print('Train: X=%s, y=%s' % (x_train.shape, y_train.shape))
     print('Test: X=%s, y=%s' % (x_test.shape, y_test.shape))
     print('Class Labels: ',np.unique(y_train))
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 2s 0us/step
170508288/170498071 [==============================] - 2s 0us/step
Train: X=(50000, 32, 32, 3), y=(50000, 1)
Test: X=(10000, 32, 32, 3), y=(10000, 1)
Class Labels:  [0 1 2 3 4 5 6 7 8 9]
```

قسمت دوم مربوط به لود دیتا هست. یک تابع به کمک تابع load_data که در داخل دیتاست cifar10 هست، تعریف می کنیم. و با استفاده از آن دیتاهای آموزش و تست را جدا کرده و در خروجی تعداد و ابعاد دیتا تست و آموزش و همچنین کلاس های یونیک موجود در تابع را می بینیم.

3. Plot a few images

+ Code + Text

```
def showImages(num_row,num_col,X,Y):

    (X_rand, Y_rand) = shuffle(X, Y)

    fig, axes = plt.subplots(num_row,num_col,figsize = (12,12))
    axes = axes.ravel()
    for i in range(0, num_row*num_col):
        axes[i].imshow(X_rand[i])
        axes[i].set_title("{}".format(labels[Y_rand.item(i)]))
        axes[i].axis('off')
        plt.subplots_adjust(wspace =1)

    return


labels = ['Airplane', 'Automobile', 'Bird', 'Cat', 'Deer', 'Dog', 'Frog', 'Horse', 'Ship', 'Truck']

num_row = 5
num_col = 5
showImages(num_row,num_col,X =x_train,Y = y_train)
```

در قسمت سوم یه تعداد از دیتاهای موجود رو به صورت رندوم همراه با برچسب آنها نمایش میدهیم. برای این کار یک تابع تعریف می کنیم و با shuffle میگیم x و y ها را رندوم انتخاب کن. ودر یک حلقه for ، تصویر مربوط به x های رندومی که انتخاب شده رو نمایش بده و از لیست labels برچسب مربوط به آن عکس را هم نمایش بده. همانطور که در قسمت دوم مشاهده کردیم برچسب های یونیک خروجی به صورت اعداد ۰ تا ۹ هستند ، که ما در اینجا با استفاده از تعریف لیست labels و دستور format این اعداد ۰ تا ۹ را به لیبل های معنادار cat , dog, deer,... تبدیل کردیم. در نهایت چه تعداد دیتا را که میخواهیم نشان داده شود در قالب تعداد سطر و ستون ها در قسمت num_row و num_col وارد می کنیم.

4. Normalize input

```
x_train=x_train.astype('float32')
x_test=x_test.astype('float32')

#normalization from [0:255] to [0:1] // scale the data to lie between 0 to 1
x_train /= 255
x_test /= 255

#convert labels to one_hot vectors
y_train=np_utils.to_categorical(y_train)
y_test=np_utils.to_categorical(y_test)
```

قسمت ۴ برای نرمال سازی دیتا هست. ورودی های ما عکس های ۳۲*۳۲ پیکسل رنگی هستند. یعنی هر کدام به صورت یک ماتریس هست که درایه های آن عددی بین ۰ تا ۲۵۵ هست و ۳ کانال RGB دارد. در نرمال سازی داده های ورودی آنها را به ۲۵۵ تقسیم میکنیم تا مقدار همه درایه های بین ۰ تا ۱ قرار گیرد.

خروجی داده های تست و آموزش را هم به صورت بردار one hotمینویسیم. یعنی فقط سطر مربوط به برچسب درست ۱ و مابقی سطرها ۰ باشد.

```
[6]  model = Sequential()
     model.add(Conv2D(32, (5,5), activation = 'relu', padding = 'same', input_shape = (32,32,3)))
     model.add(MaxPooling2D(pool_size = (2,2)))
     model.add(Conv2D(64, (5,5), activation = 'relu', padding = 'same'))
     model.add(MaxPooling2D(pool_size = (2,2)))
     model.add(Conv2D(64, (5,5), activation = 'relu', padding = 'same'))
     model.add(MaxPooling2D(pool_size = (2,2)))
     model.add(Conv2D(128, (5,5), activation = 'relu', padding = 'same'))
     model.add(MaxPooling2D(pool_size = (2,2)))
     model.add(Conv2D(128, (5,5), activation = 'relu', padding = 'same'))
     model.add(MaxPooling2D(pool_size = (2,2)))
     model.add(Conv2D(128, (5,5), activation = 'relu', padding = 'same'))
     model.add(Flatten())
     model.add(Dense(64, activation = 'relu'))
     model.add(Dense(10, activation = 'softmax'))
```

مرحله بعدی مربوط تعریف مدل است. ابتدا یک معماری با ۶ لایه کانولوشنی ۵*۵ تعریف کردیم، که بعد از هرکدام مکس پولینگ ۲*۲ زدیم و تابع فعالسازی برای همه لایه ها reluدر نظر گرفته شده.

```
=================================================================
Total params: 804,298
Trainable params: 804,298
Non-trainable params: 0
_________________________________________________________________
```

از مدلی که ساختیم summaryمیگیریم و و تعداد پارامترهای این مدل را بررسی میکنیم.

7. Compile your model

```
[8]  model.compile(loss='categorical_crossentropy', optimizer='Adam',metrics=['accuracy'])
```

8. Checkpoint

```
[9]  # Write a checkpoint which stops the training process when the val accuracy doesn't change for 5 epochs

     callback = tensorflow.keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=5)
```

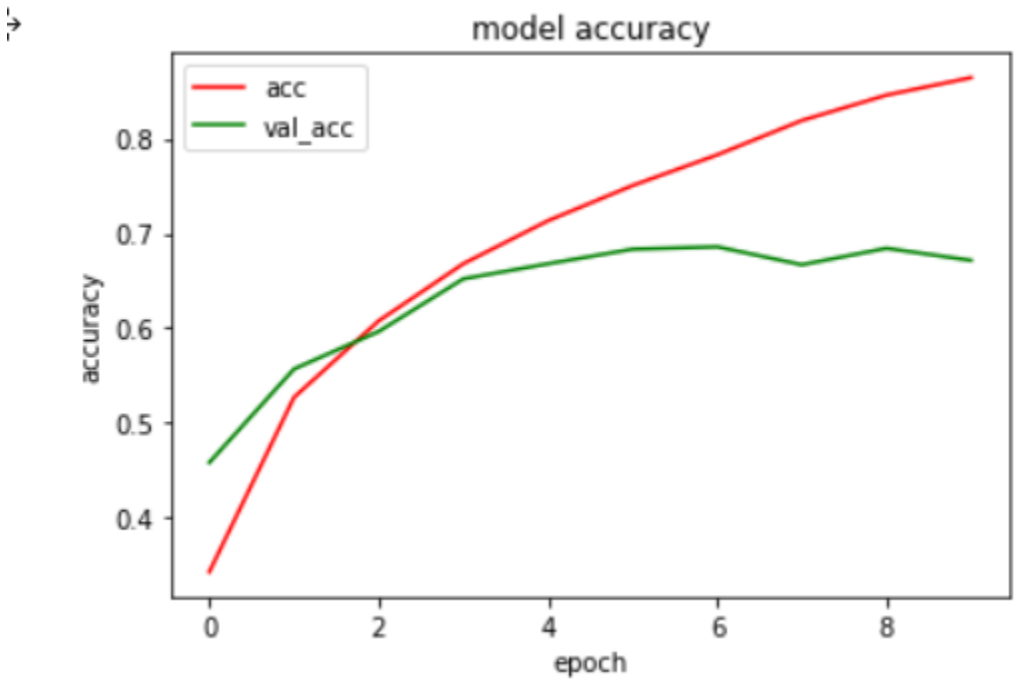
مدل را با loss ، categorical crossentropy و با اپتیمایزر Adam کامپایل میکنیم.

Callback را از نوع earlystopping تعریف کرده و میگیم اگر val acc تو ۵تا epoch تغییر نکرد، فرایند آموزش را متوقف کن.

```
[10] history =model.fit(x_train,y_train,epochs=10,validation_split=0.3,callbacks=[callback])

Epoch 1/10
1094/1094 [==============================] - 282s 257ms/step - loss: 1.7249 - accuracy: 0.3431 - val_loss: 1.4435 - val_accuracy: 0.4584
Epoch 2/10
1094/1094 [==============================] - 282s 258ms/step - loss: 1.3028 - accuracy: 0.5271 - val_loss: 1.2299 - val_accuracy: 0.5570
Epoch 3/10
1094/1094 [==============================] - 279s 255ms/step - loss: 1.1027 - accuracy: 0.6077 - val_loss: 1.1634 - val_accuracy: 0.5965
Epoch 4/10
1094/1094 [==============================] - 277s 253ms/step - loss: 0.9398 - accuracy: 0.6680 - val_loss: 1.0016 - val_accuracy: 0.6521
Epoch 5/10
1094/1094 [==============================] - 282s 258ms/step - loss: 0.8134 - accuracy: 0.7134 - val_loss: 0.9730 - val_accuracy: 0.6680
Epoch 6/10
1094/1094 [==============================] - 280s 256ms/step - loss: 0.7108 - accuracy: 0.7505 - val_loss: 0.9455 - val_accuracy: 0.6834
Epoch 7/10
1094/1094 [==============================] - 278s 254ms/step - loss: 0.6141 - accuracy: 0.7829 - val_loss: 0.9828 - val_accuracy: 0.6859
Epoch 8/10
1094/1094 [==============================] - 273s 249ms/step - loss: 0.5209 - accuracy: 0.8192 - val_loss: 1.0699 - val_accuracy: 0.6670
Epoch 9/10
1094/1094 [==============================] - 272s 249ms/step - loss: 0.4403 - accuracy: 0.8459 - val_loss: 1.1164 - val_accuracy: 0.6843
Epoch 10/10
1094/1094 [==============================] - 274s 250ms/step - loss: 0.3866 - accuracy: 0.8644 - val_loss: 1.2364 - val_accuracy: 0.6716
```

مدلی که ساختیم را روی داده های آموزش فیت میکنیم و طبق نتیجه میبینیم که دقت آن در طی ۱۰ تکرار از ۳۴٪ به ۸۶٪ رسیده.

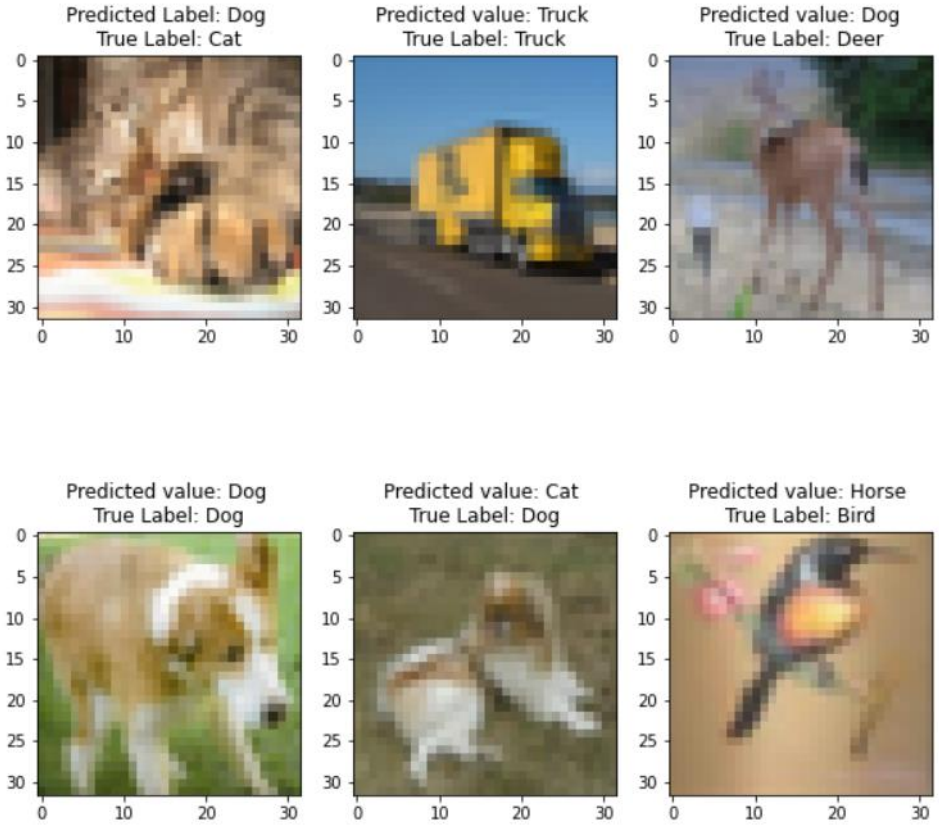


نمودار مربوط به دقت **مدل اول** (با ۶لایه کانولوشنی ۵*۵) روی داده های آموزش و داده های validation در epoch های مختلف

```
#evaluation the trained model
[test_loss, test_acc]= model.evaluate(x_test, y_test)
print('test loss=',test_loss)
print('test accuracy=', test_acc)
```

```
313/313 [==============================] - 23s 72ms/step - loss: 1.2708 - accuracy: 0.6665
test loss= 1.2708226442337036
test accuracy= 0.6664999723434448
```

در ادامه بعد از سیو کردن مدل، روی داده های تست پیش بینی انجام داده و ارزیابی میکنیم. مدل اول برای داده های تست از دقت ۶۶٪ برخوردار است. یعنی قادر است بیش از نیمی از داده های تست را درست برچسب گذاری کند. در نگاه دقیق تر دقت مدل روی دیتا آموزش ۸۶٪ بود ولی دقت روی دیتا تست کاهش پیدا کرده، میتوان گفت که مدل در حین آموزش روی داده های آموزش و ولیدیشین اندکی دچار اورفیت شده است.



در مرحله آخر تعدادی از داده های تست را نمایش داده و برچسب واقعی و برچسب پیش بینی شده آنها را مقایسه میکنیم. همانطور که مشاهده میشود از ۶تا انتخاب رندوم عکس از دیتا تست ۴تا اشتباه و ۲تا درست برچسب گذاری شده اند.

در ادامه به دنبال بررسی تاثیر عوامل مختلف روی بهبود عملکرد مدل و فرایند آموزش هستیم.

برای این منظور تعداد فیلترها و ابعاد کرنل را تغییر میدهیم :

مدل دوم:

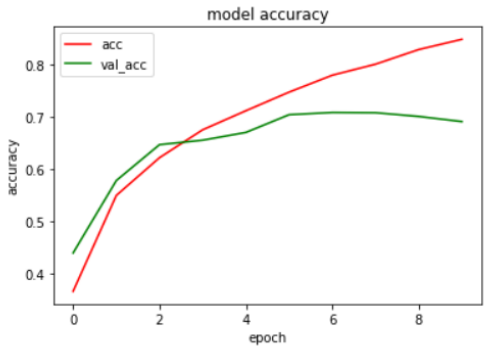
```
model = Sequential()
model.add(Conv2D(32, (3,3), activation = 'relu', padding = 'same', input_shape = (32,32,3)))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Conv2D(64, (3,3), activation = 'relu', padding = 'same'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Conv2D(64, (3,3), activation = 'relu', padding = 'same'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Conv2D(128, (3,3), activation = 'relu', padding = 'same'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Conv2D(128, (3,3), activation = 'relu', padding = 'same'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Conv2D(128, (3,3), activation = 'relu', padding = 'same'))
model.add(Flatten())
model.add(Dense(64, activation = 'relu'))
model.add(Dense(10, activation = 'softmax'))
```

مدل دوم را به این صورت تعریف کردیم. ۶ لایه کانولوشنی ۳*۳. لایه اول با ۳۲ فیلتر، لایه دوم و سوم ۶۴ فیلتر، لایه ۴ و ۵ و ۶ با ۱۲۸ فیلتر. (تعداد فیلترها مانند مدل اول است فقط سایز کرنل را تغییر دادیم).

```
history =model.fit(x_train,y_train,epochs=10,validation_split=0.3,callbacks=[callback])
```

```
Epoch 1/10
1094/1094 [==============================] - 120s 109ms/step - loss: 1.6856 - accuracy: 0.3671 - val_loss: 1.5297 - val_accuracy: 0.4403
Epoch 2/10
1094/1094 [==============================] - 117s 107ms/step - loss: 1.2463 - accuracy: 0.5503 - val_loss: 1.1686 - val_accuracy: 0.5788
Epoch 3/10
1094/1094 [==============================] - 117s 107ms/step - loss: 1.0478 - accuracy: 0.6225 - val_loss: 0.9855 - val_accuracy: 0.6474
Epoch 4/10
1094/1094 [==============================] - 117s 107ms/step - loss: 0.9108 - accuracy: 0.6759 - val_loss: 0.9774 - val_accuracy: 0.6559
Epoch 5/10
1094/1094 [==============================] - 115s 105ms/step - loss: 0.8052 - accuracy: 0.7123 - val_loss: 0.9554 - val_accuracy: 0.6708
Epoch 6/10
1094/1094 [==============================] - 120s 110ms/step - loss: 0.7083 - accuracy: 0.7478 - val_loss: 0.8645 - val_accuracy: 0.7047
Epoch 7/10
1094/1094 [==============================] - 116s 106ms/step - loss: 0.6259 - accuracy: 0.7801 - val_loss: 0.8742 - val_accuracy: 0.7089
Epoch 8/10
1094/1094 [==============================] - 117s 107ms/step - loss: 0.5635 - accuracy: 0.8012 - val_loss: 0.8912 - val_accuracy: 0.7085
Epoch 9/10
1094/1094 [==============================] - 115s 105ms/step - loss: 0.4886 - accuracy: 0.8292 - val_loss: 0.9960 - val_accuracy: 0.7013
Epoch 10/10
1094/1094 [==============================] - 117s 107ms/step - loss: 0.4356 - accuracy: 0.8488 - val_loss: 1.0079 - val_accuracy: 0.6915
```

```
=================================
Total params: 434,250
Trainable params: 434,250
Non-trainable params: 0
_________________________________
```




```
313/313 [==============================] - 7s 21ms/step - loss: 1.0277 - accuracy: 0.6809
test loss= 1.027711033821106
test accuracy= 0.680899977684021
```

نتیجه: کاهش ابعاد کرنل ها تغییر محسوسی در دقت داده های تست و آموزش به وجود نمی اورد. با توجه به نمودار، میزان دقت در داده های ولیدیشین به دقت داده های آموزش نسبت به مدل اول همگرا تر است. تعداد پارامتر ها با کاهش ابعاد کرنل کاهش می یابد.

مدل سوم:

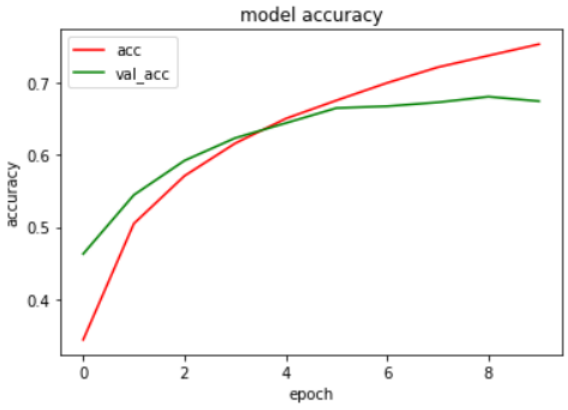
۶تا کانولوشن ۳*۳.
۳تا اولی با ۳۲ فیلتر و
۳تا دومی با ۶۴ فیلتر.
نسبت به مدل قبل
تعداد فیلتر هارا
کاهش دادیم.

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

```
Epoch 1/10
1094/1094 [==============================] - 68s 61ms/step - loss: 1.7470 - accuracy: 0.3442 - val_loss: 1.4529 - val_accuracy: 0.4630
Epoch 2/10
1094/1094 [==============================] - 65s 60ms/step - loss: 1.3546 - accuracy: 0.5049 - val_loss: 1.2672 - val_accuracy: 0.5446
Epoch 3/10
1094/1094 [==============================] - 65s 59ms/step - loss: 1.1833 - accuracy: 0.5710 - val_loss: 1.1575 - val_accuracy: 0.5922
Epoch 4/10
1094/1094 [==============================] - 65s 60ms/step - loss: 1.0698 - accuracy: 0.6161 - val_loss: 1.0709 - val_accuracy: 0.6234
Epoch 5/10
1094/1094 [==============================] - 66s 61ms/step - loss: 0.9793 - accuracy: 0.6503 - val_loss: 1.0128 - val_accuracy: 0.6441
Epoch 6/10
1094/1094 [==============================] - 65s 59ms/step - loss: 0.9084 - accuracy: 0.6755 - val_loss: 0.9705 - val_accuracy: 0.6649
Epoch 7/10
1094/1094 [==============================] - 66s 60ms/step - loss: 0.8443 - accuracy: 0.6997 - val_loss: 0.9579 - val_accuracy: 0.6675
Epoch 8/10
1094/1094 [==============================] - 66s 60ms/step - loss: 0.7885 - accuracy: 0.7214 - val_loss: 0.9516 - val_accuracy: 0.6728
Epoch 9/10
1094/1094 [==============================] - 65s 59ms/step - loss: 0.7385 - accuracy: 0.7375 - val_loss: 0.9428 - val_accuracy: 0.6807
Epoch 10/10
1094/1094 [==============================] - 63s 57ms/step - loss: 0.6947 - accuracy: 0.7533 - val_loss: 0.9544 - val_accuracy: 0.6744
```

```
313/313 [==============================
test loss= 0.9799600839614868
test accuracy= 0.6657999753952026
```

```
Total params: 116,554
Trainable params: 116,554
Non-trainable params: 0
```



نتیجه: کاهش تعداد فیلترها باعث کم شدن تعداد پارامترها و همچنین دقت روی داده های آموزش می شود.

در این مدل دقت داده های آموزش و ولیدیشن نسبت به ۲ مدل قبلی بهم نزدیکتر است.

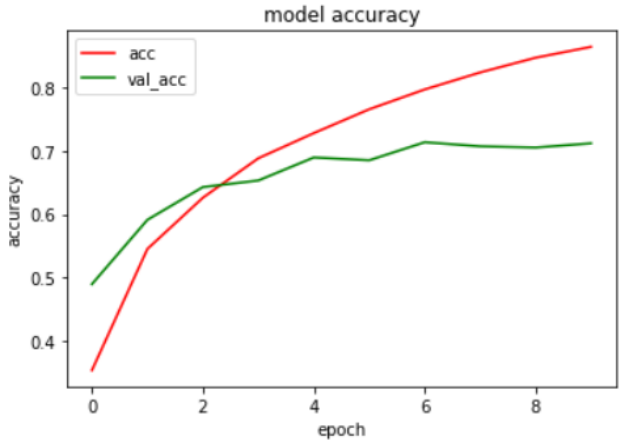
مدل ۴:

تعداد فیلترها را افزایش دادیم. با ابعاد کرنل مثل مدل قبلی.

```
model = Sequential()
model.add(Conv2D(64, (3,3), activation = 'relu', padding = 'same', input_shape = (32,32,3)))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Conv2D(64, (3,3), activation = 'relu', padding = 'same'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Conv2D(128, (3,3), activation = 'relu', padding = 'same'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Conv2D(128, (3,3), activation = 'relu', padding = 'same'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Conv2D(128, (3,3), activation = 'relu', padding = 'same'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Conv2D(128, (3,3), activation = 'relu', padding = 'same'))
model.add(Flatten())
model.add(Dense(64, activation = 'relu'))
model.add(Dense(10, activation = 'softmax'))
```

```
Epoch 1/10
1094/1094 [==============================] - 184s 168ms/step - loss: 1.7046 - accuracy: 0.3539 - val_loss: 1.3955 - val_accuracy: 0.4898
Epoch 2/10
1094/1094 [==============================] - 184s 168ms/step - loss: 1.2461 - accuracy: 0.5454 - val_loss: 1.1316 - val_accuracy: 0.5913
Epoch 3/10
1094/1094 [==============================] - 185s 169ms/step - loss: 1.0355 - accuracy: 0.6262 - val_loss: 1.0201 - val_accuracy: 0.6430
Epoch 4/10
1094/1094 [==============================] - 185s 169ms/step - loss: 0.8798 - accuracy: 0.6885 - val_loss: 1.0169 - val_accuracy: 0.6534
Epoch 5/10
1094/1094 [==============================] - 184s 168ms/step - loss: 0.7734 - accuracy: 0.7282 - val_loss: 0.8921 - val_accuracy: 0.6897
Epoch 6/10
1094/1094 [==============================] - 181s 166ms/step - loss: 0.6644 - accuracy: 0.7659 - val_loss: 0.9318 - val_accuracy: 0.6853
Epoch 7/10
1094/1094 [==============================] - 176s 161ms/step - loss: 0.5833 - accuracy: 0.7971 - val_loss: 0.8777 - val_accuracy: 0.7138
Epoch 8/10
1094/1094 [==============================] - 177s 162ms/step - loss: 0.5037 - accuracy: 0.8240 - val_loss: 0.9437 - val_accuracy: 0.7074
Epoch 9/10
1094/1094 [==============================] - 177s 162ms/step - loss: 0.4403 - accuracy: 0.8473 - val_loss: 0.9399 - val_accuracy: 0.7055
Epoch 10/10
1094/1094 [==============================] - 177s 162ms/step - loss: 0.3847 - accuracy: 0.8644 - val_loss: 1.0035 - val_accuracy: 0.7121
```

```
313/313 [==============================
test loss= 1.029495120048523
test accuracy= 0.7046999931335449
```




```
Total params: 564,234
Trainable params: 564,234
Non-trainable params: 0
_________________________________________________________________
```

نتیجه: این مدل با دقت آموزش ۸۶٪ و دقت تست ۷۰٪ بالاترین دقتها را در بین ۴ مدل دارد. یعنی افزایش تعداد فیلتر ها باعث افزایش تعداد پارامترها و به دنبال آن پیچیدگی مدل میشود و دقت و فرایند آموزش را بهبود می بخشد. مدل ۴ را به عنوان بهترین مدل تغییر تعداد فیلتر و کرنل برداشته و در مدل۵ تابع فعالسازی آنرا تغییر میدهیم.

مدل ۵:

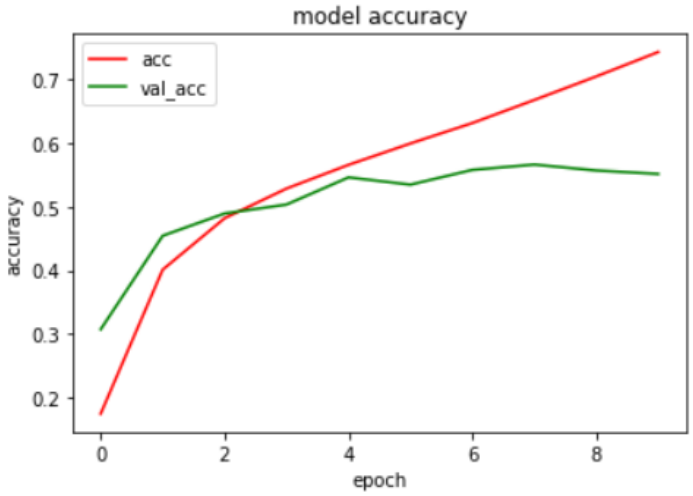
به جای relu از تابع فعالسازی softplus استفاده میکنیم.

```
model = Sequential()
model.add(Conv2D(64, (3,3), activation = 'softplus', padding = 'same', input_shape = (32,32,3)))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Conv2D(64, (3,3), activation = 'softplus', padding = 'same'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Conv2D(128, (3,3), activation = 'softplus', padding = 'same'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Conv2D(128, (3,3), activation = 'softplus', padding = 'same'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Conv2D(128, (3,3), activation = 'softplus', padding = 'same'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Conv2D(128, (3,3), activation = 'softplus', padding = 'same'))
model.add(Flatten())
model.add(Dense(64, activation = 'softplus'))
model.add(Dense(10, activation = 'softmax'))
```

```
Epoch 1/10
1094/1094 [==============================] - 214s 195ms/step - loss: 2.1731 - accuracy: 0.1751 - val_loss: 1.8847 - val_accuracy: 0.3075
Epoch 2/10
1094/1094 [==============================] - 210s 192ms/step - loss: 1.6450 - accuracy: 0.4011 - val_loss: 1.5030 - val_accuracy: 0.4543
Epoch 3/10
1094/1094 [==============================] - 211s 193ms/step - loss: 1.4346 - accuracy: 0.4819 - val_loss: 1.4304 - val_accuracy: 0.4893
Epoch 4/10
1094/1094 [==============================] - 210s 192ms/step - loss: 1.3132 - accuracy: 0.5285 - val_loss: 1.3802 - val_accuracy: 0.5034
Epoch 5/10
1094/1094 [==============================] - 210s 192ms/step - loss: 1.2141 - accuracy: 0.5657 - val_loss: 1.2821 - val_accuracy: 0.5460
Epoch 6/10
1094/1094 [==============================] - 211s 193ms/step - loss: 1.1231 - accuracy: 0.5993 - val_loss: 1.3415 - val_accuracy: 0.5346
Epoch 7/10
1094/1094 [==============================] - 211s 193ms/step - loss: 1.0322 - accuracy: 0.6313 - val_loss: 1.2853 - val_accuracy: 0.5578
Epoch 8/10
1094/1094 [==============================] - 211s 193ms/step - loss: 0.9286 - accuracy: 0.6675 - val_loss: 1.3190 - val_accuracy: 0.5661
Epoch 9/10
1094/1094 [==============================] - 210s 192ms/step - loss: 0.8229 - accuracy: 0.7045 - val_loss: 1.3942 - val_accuracy: 0.5568
Epoch 10/10
1094/1094 [==============================] - 211s 193ms/step - loss: 0.7176 - accuracy: 0.7428 - val_loss: 1.5466 - val_accuracy: 0.5514
```

```
_________________________________________________________________
Total params: 564,234
Trainable params: 564,234
Non-trainable params: 0
_________________________________________________________________
```

```
test loss= 1.5401489734649658
test accuracy= 0.5503000020980835
```



نتیجه: دقت به ۷۴ ٪ کاهش یافت. تابع فعالساز relu عملکرد قویتری دارد.

در این مدل دقت از ۱۷ درصد با شیب تقریبا زیادی افزایش پیدا کرد و به ۷۴ درصد که احتمال اورفیت شدن را افزایش میدهد. همانطور که میبینیم دقت تست ۵۵درصد است که مقدار قابل قبولی نیست.

مدل۶:

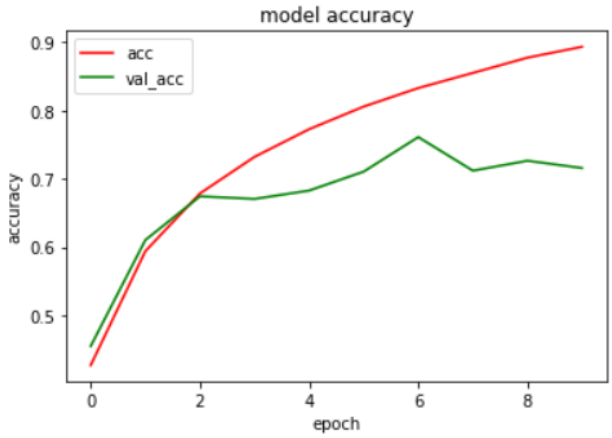
بین لایه های کانولوشن، batchnormalization و بین لایه های FC ، dropout اضافه کردیم.

```
model = Sequential()
model.add(Conv2D(64, (3,3), activation = 'relu', padding = 'same', input_shape = (32,32,3)))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(BatchNormalization())
model.add(Conv2D(64, (3,3), activation = 'relu', padding = 'same'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(BatchNormalization())
model.add(Conv2D(128, (3,3), activation = 'relu', padding = 'same'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(BatchNormalization())
model.add(Conv2D(128, (3,3), activation = 'relu', padding = 'same'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(BatchNormalization())
model.add(Conv2D(128, (3,3), activation = 'relu', padding = 'same'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(BatchNormalization())
model.add(Conv2D(128, (3,3), activation = 'relu', padding = 'same'))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(64, activation = 'relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation = 'softmax'))
```

```
Epoch 1/10
1094/1094 [==============================] - 193s 176ms/step - loss: 1.5989 - accuracy: 0.4271 - val_loss: 1.7456 - val_accuracy: 0.4551
Epoch 2/10
1094/1094 [==============================] - 188s 172ms/step - loss: 1.1834 - accuracy: 0.5941 - val_loss: 1.0857 - val_accuracy: 0.6104
Epoch 3/10
1094/1094 [==============================] - 190s 173ms/step - loss: 0.9658 - accuracy: 0.6786 - val_loss: 0.9601 - val_accuracy: 0.6745
Epoch 4/10
1094/1094 [==============================] - 190s 173ms/step - loss: 0.8143 - accuracy: 0.7323 - val_loss: 0.9766 - val_accuracy: 0.6707
Epoch 5/10
1094/1094 [==============================] - 187s 171ms/step - loss: 0.6989 - accuracy: 0.7728 - val_loss: 0.9305 - val_accuracy: 0.6829
Epoch 6/10
1094/1094 [==============================] - 188s 172ms/step - loss: 0.5977 - accuracy: 0.8059 - val_loss: 0.9047 - val_accuracy: 0.7107
Epoch 7/10
1094/1094 [==============================] - 189s 172ms/step - loss: 0.5230 - accuracy: 0.8331 - val_loss: 0.7740 - val_accuracy: 0.7613
Epoch 8/10
1094/1094 [==============================] - 189s 173ms/step - loss: 0.4515 - accuracy: 0.8552 - val_loss: 0.9642 - val_accuracy: 0.7121
Epoch 9/10
1094/1094 [==============================] - 190s 173ms/step - loss: 0.3831 - accuracy: 0.8776 - val_loss: 1.0319 - val_accuracy: 0.7265
Epoch 10/10
1094/1094 [==============================] - 188s 172ms/step - loss: 0.3391 - accuracy: 0.8935 - val_loss: 1.1111 - val_accuracy: 0.7159
```

```
=================================
Total params: 566,282
Trainable params: 565,258
Non-trainable params: 1,024
_________________________________
```

```
313/313 [==============================
test loss= 1.1273313760757446
test accuracy= 0.7078999876976013
```

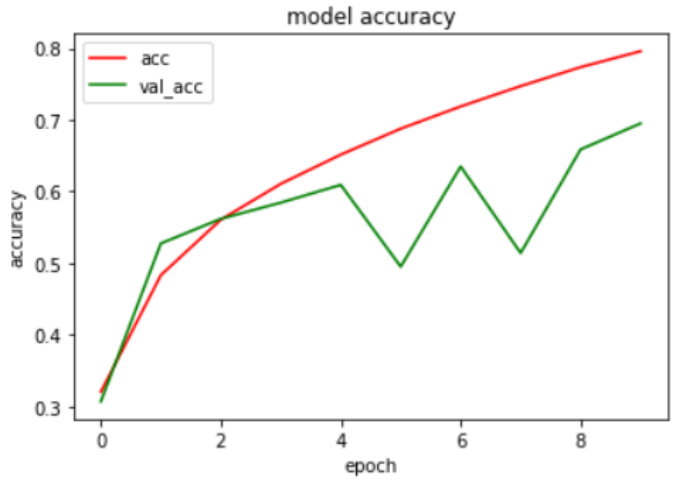


نتیجه: افزودن لایه های بچ نرمالیزیشن و دراپ اوت باعث بوجود امدن پارامترهای غیر قابل آموزش در مدل شد. و در حالت کلی دقت آموزش را به بالاترین میزان در بین مدل های بالا رساند. پس افزودن این لایه ها باعث بهبود عملکرد می شود.

مدل ۷: در این مدل از همان لایه های مدل ۶ که بهترین عملکرد را در بین مدل های قبلی داشت استفاده کردیم. فقط در قسمت کامپایل کردن مدل از اپتیمایزر SGD(stochastic gradient descent) استفاده کردیم.

```
model.compile(loss='categorical_crossentropy', optimizer='SGD',metrics=['accuracy'])
```

```
Epoch 1/10
1094/1094 [==============================] - 198s 180ms/step - loss: 1.8400 - accuracy: 0.3202 - val_loss: 1.8951 - val_accuracy: 0.3068
Epoch 2/10
1094/1094 [==============================] - 195s 178ms/step - loss: 1.4489 - accuracy: 0.4830 - val_loss: 1.3460 - val_accuracy: 0.5273
Epoch 3/10
1094/1094 [==============================] - 187s 171ms/step - loss: 1.2654 - accuracy: 0.5605 - val_loss: 1.2221 - val_accuracy: 0.5617
Epoch 4/10
1094/1094 [==============================] - 191s 174ms/step - loss: 1.1275 - accuracy: 0.6106 - val_loss: 1.1710 - val_accuracy: 0.5843
Epoch 5/10
1094/1094 [==============================] - 193s 177ms/step - loss: 1.0145 - accuracy: 0.6513 - val_loss: 1.1191 - val_accuracy: 0.6091
Epoch 6/10
1094/1094 [==============================] - 195s 179ms/step - loss: 0.9278 - accuracy: 0.6876 - val_loss: 1.5755 - val_accuracy: 0.4949
Epoch 7/10
1094/1094 [==============================] - 195s 178ms/step - loss: 0.8372 - accuracy: 0.7187 - val_loss: 1.1349 - val_accuracy: 0.6348
Epoch 8/10
1094/1094 [==============================] - 192s 175ms/step - loss: 0.7563 - accuracy: 0.7471 - val_loss: 1.7290 - val_accuracy: 0.5139
Epoch 9/10
1094/1094 [==============================] - 194s 177ms/step - loss: 0.6841 - accuracy: 0.7735 - val_loss: 1.0596 - val_accuracy: 0.6587
Epoch 10/10
1094/1094 [==============================] - 195s 178ms/step - loss: 0.6165 - accuracy: 0.7958 - val_loss: 0.9396 - val_accuracy: 0.6950
```



نتیجه: این اپتیمایزر دقت کمتری نسبت به آدام دارد . چون درصد دقت آن همانطور که مشاهده میشود کمتر شد.

مدل ۶ را به عنوان بهترین مدل انتخاب کرده و Data Augmentation را روی آن اعمال میکنیم.

مدل نهایی:

در مدل آخر از داده افزایی استفاده میکنیم.

```
batch_size = 32
epochs = 10

# create a copy of your model and train it with augmented data
model2 = tensorflow.keras.models.clone_model(model)

model2.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])


# Define transformations for train data
datagen = ImageDataGenerator(
        width_shift_range=0.1,  # randomly shift images horizontally
        height_shift_range=0.1,
        horizontal_flip=True,
        vertical_flip=False)

# Fit the model on the batches generated by datagen.flow().
history2 = model2.fit(datagen.flow(x_train, y_train, batch_size=batch_size),
                      steps_per_epoch=int(np.ceil(x_train.shape[0] / float(batch_size))),
                      epochs=epochs,
                      validation_data=(x_test, y_test),
                      workers=4
          )

model2.evaluate(x_test, y_test)
```

```
Epoch 1/10
1563/1563 [==============================] - 283s 180ms/step - loss: 1.5962 - accuracy: 0.4282 - val_loss: 1.3441 - val_accuracy: 0.5133
Epoch 2/10
1563/1563 [==============================] - 280s 179ms/step - loss: 1.2031 - accuracy: 0.5900 - val_loss: 1.1069 - val_accuracy: 0.6183
Epoch 3/10
1563/1563 [==============================] - 282s 181ms/step - loss: 1.0319 - accuracy: 0.6572 - val_loss: 0.8755 - val_accuracy: 0.7003
Epoch 4/10
1563/1563 [==============================] - 279s 178ms/step - loss: 0.9281 - accuracy: 0.6943 - val_loss: 0.7543 - val_accuracy: 0.7434
Epoch 5/10
1563/1563 [==============================] - 277s 177ms/step - loss: 0.8581 - accuracy: 0.7189 - val_loss: 0.9221 - val_accuracy: 0.6896
Epoch 6/10
1563/1563 [==============================] - 278s 178ms/step - loss: 0.8159 - accuracy: 0.7351 - val_loss: 0.9155 - val_accuracy: 0.6928
Epoch 7/10
1563/1563 [==============================] - 279s 178ms/step - loss: 0.7661 - accuracy: 0.7518 - val_loss: 0.7703 - val_accuracy: 0.7410
Epoch 8/10
1563/1563 [==============================] - 279s 179ms/step - loss: 0.7295 - accuracy: 0.7654 - val_loss: 0.7018 - val_accuracy: 0.7667
Epoch 9/10
1563/1563 [==============================] - 279s 178ms/step - loss: 0.7069 - accuracy: 0.7721 - val_loss: 0.6587 - val_accuracy: 0.7790
Epoch 10/10
1563/1563 [==============================] - 278s 178ms/step - loss: 0.6862 - accuracy: 0.7778 - val_loss: 0.6564 - val_accuracy: 0.7825
313/313 [==============================] - 11s 34ms/step - loss: 0.6564 - accuracy: 0.7825
[0.656436562538147, 0.7825000286102295]
```