تشریحی:

سوال ۱)

برای regression Problem داریم:

۱: Mean Absolute Error یا MAE

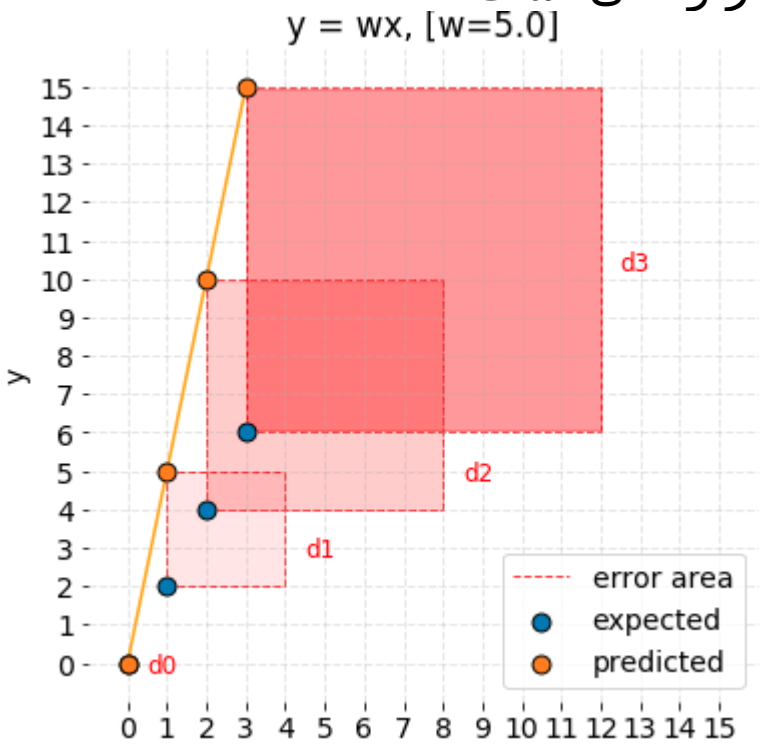
این تابع برای خطا میانگین اندازه ی خطاها را را برای گروهی از داده‌ها حساب می‌کند درواقع آن میانگین قدر مطلق تفاوت میان مقادیری که توسط شبکه حدس زده شده و مقادیر واقعی آن را به عنوان خطا حساب می‌کند

فرمول آن به شکل زیر است:

$$MAE = \frac{1}{m}\sum_{i=1}^{m}|\hat{y}^{(i)} - y^{(i)}|$$

۲: Mean Square Error یا MSE

این تابع برای مقادیر خطا میانگین توان دو ی اختلاف مقادیر حدس زده شده و مقادیر واقعی را به عنوان خطا به ما می‌دهد درواقع مانند همان MAE است که به جای قدر مطلق از توان ۲ استفاده شده است. درواقع خطا برابر مساحت یک مربع به ضلع اختلاف تا مقدار واقعی است



برای Classification Problem:

1:Cross-entropy

از این تابع برای مسایل طبقه بندی استفاده می شود. این تابع برای جاهایی که خروجی احتمال بین ۰ تا ۱ دارد استفاده می شود. به این صورت است که برای هر کلاس یک عدد به عنوان خطا به صورت زیر پیدا می‌کند

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$

که m تعداد کلاس‌ها و p احتمال بودن آن کلاس y مقدار درست آن کلاس برای آن خروجی است که یا ۰ یا ۱ است و اینگونه خطا را محاسبه می‌کند

2:Hinge

این تابع برای مسایل طبقه بندی استفاده می شه. و این‌گونه است که برای هر کلاس ماکسیموم بین عدد ۰ و یا 1- y * t را بر میگرداند که اگر با احتمال خوبی درست جواب داده باشد دیگر ۰ نمی‌شود که م عنی خطا نداشتن است

سؤال ۲)

هر تابع بهینه سازی که حداقل به یک مشتق از مرتبه اول آن نیاز دارد بهینه ساز مرتبه اول است و هر تابعی که مشتق مرتبه دوم بخواهد و اول در آن نباشد مرتبه دوم است که به آن هسیان نیز می‌گویند بیشترین استفاده از مرتبه اول gradient decent است که به ما کم شدن یا زیاد شدن تابع در یک نقطه را می دهد. تابع مرتبه دوم loss function را مینیموم یا ماکسیموم می‌کند و چون محاسبه مرتبه دوم وقت گیر است معمولاً از آن استفاده نمی‌شود

تفاوت ها:

تابع های مرتبه اول محاسبه ساده دارند و سریع اند یعنی زمان کمتری مصرف می‌کنند و با دیتاست های بزرگ سریع‌تر همگرا می‌شوند .

برتبه دوم فقط وقتی سریع‌تر است که مشتق مرتبه دمو تابع را بدانیم در غیر این صورت این توابع همیشه کند تر و مموری بیشتری نیاز دارند
مثال از مرتبه اول مانند Adam, first order gradient decent
و از مرتبه دوم مانند second order gradient deecnt

سؤال ۳)

overfitt به معنی آن است که شبکه طراحی شده برای داده‌های دیده شده (داده‌های اموزش) خوب عمل می‌کند ولی برای داده‌هایی که ندیده (تست و ولیدیشن) دقت مناسبی ندارد. دلیل آن نویز و اعوجاج داده هاست در این حالت شبکه نتوانسته به اندازه کافی برای تمام داده‌های دیگر generalize کند و در این حالت شبکه دقت پایینی دارد. برای رفع آن ۳ مورد آورده شده است:
۱) استفاده از Data Augmentaion: با استفاده از داده‌های augment شده می‌توانیم نوییز و اعوجاج داده‌ها را به گونه‌ای تغییر دهیم دا شبکه بتواند برای باقی داده‌ها به جز داده‌های آموزش نیز پیشبینی خوبی داشته باشد
۲)یک راه دیگر برای جلو گیری از آن ساده کردن شبکه است: وقتی شبکه overfit می‌شود می توانییم شبکه را با کم کردن لایه ها یا کم کردن تعداد نورون و یا فیلتر در هر لایه شبکه را ساده‌تر کنیم تا شبکه از overfit شدن جلوگیری شود. که البته هیچ قاعده ای برای اینکه چقدر می‌توان شبکه را کوچک یا بزرگ کرد وجود ندارد و لی اگر شبکه overfit بود می‌تواند آن را کوچک‌تر نیز امتحان کرد
۳) یک روش دیگر استفاده از Drop out است: با استفاده از آن شبکه در هر ست برای آموزش داده یک سری نورون را حذف می‌کند و آن‌ها را در نظر نمی‌گیرد با اینکار شبکه هر سری می‌تواند با یک ست از نورون ها آموزش ببیند و برای داده‌هایی که ندیده بهتر عمل کند

سؤال ۴)

توابع خطی در شبکه‌های عمیق ۲ مشکل اصلی دارند:
۱) مشتق مرتبه اول این توابع با ورودی رابطه ندارد و ثابت است. این باعث می‌شود در شبکه‌های خطی back propagation نتوان استفاده کرد و دیگر نمی‌تواند فهمید وزن ها ی لایه ها ی قبل چگونه باید باشند تا جواب بهتری حاصل شود
۲) مشکل دوم آن این است که تابع خطی باعث می‌شود شبکه شبیه آن شود که انگار فقط یک لایه داریم چرا که اگر همه ی اکتیویشن ها خطی باشند چند تابع خطی توی هم نیز باز هم خطی است پس لایه آخر با لایه اول یک رابطه خطی دارد که انگار کلاً همین یک لایه داریم و شبکه دیگر قدرت کافی برای تجزیه داده‌هایی که ندیده و یا اعوجاج دارد ندارد
برای تابع های مختلف داریم:
۱) Tanh:
Advantage:
۱)مرکزیت ۰ دارد خروجی بین -۱ تا ۱ می‌دهد که یعنی می‌تواند با آن ورودی خیلی منفی یا خیلی مثبت را به درستی خروجی به دهد
۲)تابع smooth است که یعنی خروجی برش ندارد
Disadvantage:
۱)محاسبه ی آن سنگین و وقت گیر است
۲)vanishing gradient دارد یعنی برای مقادیر خیلی زیاد یا کم ممکن است به خوبی آموزش نبیند و تغییری در خروجی ایجاد نکند

۲) ReLU:
Advantage:
۱) محاسبات آن کم است که باعث می‌شود شبکه سریع‌تر همگرا شود
۲) تابعی غیر خطی است که یعنی مشکلات خطی را ندارد
Disadvantage:
۱) وقتی ورودی ها نزدیک ۰ یا منفی هستند اکتیویشن فانکشن خروجی ۰ می‌دهد و دیگر امکان backpropagation وجود ندارد

۳) Softmax
Advantage:

۱) توانایی هندل کردن چند مدل کلاس را دارد. می‌تواند خروجی بین ۰ یا ۱ یعنی نرمال شده به دهد و خروجی کلاس‌ها را از هم جدا کند
۲)برای نورون های لایه خروجی که می‌خواهند کلاس‌ها را تشخیص دهند مناسب است
Disadvantage:
۱) استفاده از آن در لایه‌های میانی درست نیست زیرا خروجی را خراب می‌کند و backpropagation ندارد

سؤال ۵)

data augmentation یک روش برای ایجاد تنوع و ایجاد ختلاف فاصله بین داده‌ها بدون نیاز به داده جدید است. با این کار که برای داده‌های آموزش استفاده می‌شود می‌توان یک سری داده جدید از روی داده‌های قبل با ایجاد یک سری تغییرات مانند شیفت یا زوم و ... به وجود می‌آید این کار باعث می‌شود شبکه بتواند به یک سری داده که ممکن است با واقعیت اختلاف داشته باشد آموزش ببیند که در این صورت می‌تواند برای داده‌هایی که ندیده بهتر عمل کند و بتواند بهتر generalize کند. از این روش در شبکه‌های عمیق استفاده می‌شود تا بتوانند با یک سری داده برای کل داده‌ها شبکه ی بهتری را آموزش دهند.

سؤال ۶)

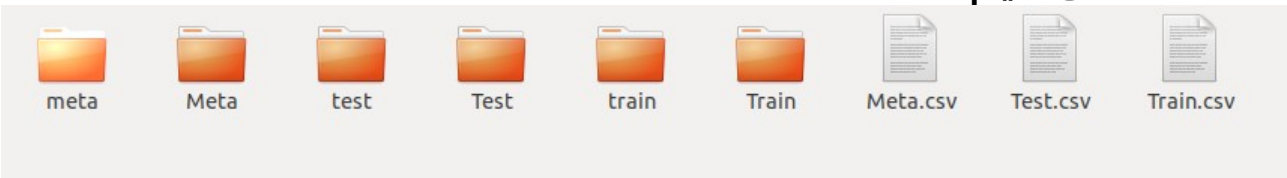
در شبکه‌ها ی عمیق اپدیت کردن وزن ها از آخر شروع شده و به عقب بر می‌گردد که یعنی شبکه در آموزش همیشه در حال عوض کردن این وزن ها است. Batch normalization یک راه برای استاندارد کردن این وزن ها برای هر mini-batch است. این اپدیت شدن‌ها ممکن است زمان زیاد ببرد و همین‌طور ممکن است به علت نرمال نشده وزن ها به مقادیر اولیه آن‌ها بسیار وابسته باشد و با یک مقدار نامناسب برای وزن ها آموزش نبیند. Batch

normalization متغیر های ورودی activation function ها را به ازای هر mini-batch نذمال می‌کند تا یک میانگین ۰ و واریانس ۱ داشته باشند تا وزن ها بتوانند به درستی و همیشه در پروسه ی اپدیت به درستی اپدیت شوند. این مار سرعت آموزش شبکه را به شدت کم می‌کند و از تأثیر مقادیر اولیه برای وزن ها می کاهد. این یک روش برای دوباره عدد گزاری وزن هاست تا همیشه بتوانند در سریع‌ترین حالت به جواب نزدیک‌تر شوند.

مقدمه:

در این پروژه می‌خواهیم یک سری داده از عکس‌های تابلو های راهنمایی و رانندگی را با یک شبکه عصبی اکوزش دهیم و تأثیر لایه ها و تابع های مختلف را روی آن ببینیم.

ابتدا داده‌های مورد نظر را از kaggle دانلود می‌کنیم این داده به صورت فایل‌های برای آموزش و تست داده هست که در ۴۳ کلاس مختلف قرار دارند
یک فایل .csv درون فایل وجود دارد که از آن برای خواندن داده‌ها استفاده می‌کنیم



با استفاده از فایل اکسل موجود داده‌ها را خوانده و به صورت x_train y_train و همین‌طور برای تست جدا می‌کنیم

این کار را به شکل زیر انجام می‌دهیم

In [4]:
```
def load_data(dataset, csv):
    images = []
    classes = []
    rows = pd.read_csv(dataset + csv)
    rows = rows.sample(frac=1).reset_index(drop=True)

    for i, row in rows.iterrows():
        img_class = row["ClassId"]
        img_path = row["Path"]
        image = os.path.join(dataset, img_path)
        image = cv2.imread(image)
        image_rs = cv2.resize(image, (30, 30), 3)

        R, G, B = cv2.split(image_rs)

        img_r = cv2.equalizeHist(R)
        img_g = cv2.equalizeHist(G)
        img_b = cv2.equalizeHist(B)

        new_image = cv2.merge((img_r, img_g, img_b))

        if i % 500 == 0:
            print(f"loaded: {i}")
        images.append(new_image)
        classes.append(img_class)

    X = np.array(images)
    y = np.array(classes)
    return X, y
```

و سپس با استفاده از آن داده‌ها را می‌خوانی

In [5]:
```
train_data = r"/home/sspc/Desktop/gtsrb-german-traffic-sign"
test_data = r"/home/sspc/Desktop/gtsrb-german-traffic-sign"
(train_X, train_Y) = load_data(train_data, "/Train.csv")
(test_X, test_Y) = load_data(test_data, "/Test.csv")
```

```
loaded: 35000
loaded: 35500
loaded: 36000
loaded: 36500
loaded: 37000
loaded: 37500
loaded: 38000
loaded: 38500
loaded: 39000
loaded: 0
loaded: 500
loaded: 1000
loaded: 1500
loaded: 2000
loaded: 2500
loaded: 3000
loaded: 3500
loaded: 4000
loaded: 4500
loaded: 5000
```

سپس بعد از خواندن داده‌ها باید آن‌ها را پیش پردازش کنیم برای اینکار ابتدا باید عکس‌های خوانده شده را نرمال سازی کنیم زیرا مقادیر آن بین ۰ تا ۲۵۵ هستند که این مقدار های زیاد می‌تواند در شبکه اختلال ایجاد کند.
کار دیگر برای پیش پردازش آن است که طبقه ها را one hot کنیم تا بتوانیم کلاس‌های مختلف را کامل از هم جدا سازیم
این کار ها را برای هر دو سری داده ی تست و آموزش انجام می دهیم:

In [6]:
```
print("UPDATE: Normalizing data")
trainX = train_X.astype("float64") / 255.0
testX = test_X.astype("float64") / 255.0
print("UPDATE: One-Hot Encoding data")
num_labels = len(np.unique(train_Y))
trainY = to_categorical(train_Y)
testY = to_categorical(test_Y)

class_totals = trainY.sum(axis=0)
class_weight = class_totals.max() / class_totals
```

```
UPDATE: Normalizing data
UPDATE: One-Hot Encoding data
```

سپس باید شبکه ای را بسازیم و داده‌ها را با آن شبکه آموزش و تست کنیم:

الف)

ابتدا شبکه را در بهترین حالت و دقت خود آموزش می‌دهیم سپس در قسمت‌های بعدی تأثیر پارامترها را روی آن می‌بینیم
برای این قسمت شبکه ما به شکل زیر است:

In [3]:

```
class RoadSignClassifier:
    def createCNN(width, height, depth, classes):
        model = Sequential()
        inputShape = (height, width, depth)
        model.add(Conv2D(filters=8, kernel_size=(5, 5), input_shape=inputShape, activation="relu"))
        model.add(MaxPooling2D(pool_size=(2, 2)))

        model.add(Conv2D(filters=16, kernel_size=(3, 3), activation="relu"))
        model.add(BatchNormalization())
        model.add(Conv2D(filters=16, kernel_size=(3, 3), activation="relu"))
        model.add(BatchNormalization())
        model.add(MaxPooling2D(pool_size=(2, 2)))

        model.add(Conv2D(filters=32, kernel_size=(3, 3), padding="same", activation="relu"))
        model.add(BatchNormalization())
        model.add(Conv2D(filters=32, kernel_size=(3, 3), padding="same", activation="relu"))
        model.add(BatchNormalization())

        model.add(Flatten())
        model.add(Dropout(0.5))
        model.add(Dense(512, activation="relu"))
        model.add(Dense(classes, activation="softmax"))
        return model
```

این شبکه از ۳ لایه کانولوشنال و ۲ لایه مخفی fully connected تشکیل شده است و از تابع relu برای اکتیویشن استفاده شده است

stride برای کانولوشن ها ۱ است تابع های آن‌ها relu و تعداد فیلتر ها به ترتیب: ۸ ۱۶ ۱۶ ۳۲ ۳۲ است
در لایه‌های مخفی اولی ۵۱۲ لایه و دومی که برای خروجی است به تعداد کلاس‌ها یعنی ۴۳ نورون قرار داده شده است
این شبکه را با مشخصات زیر آموزش می دهیم:

In [8]:
```
learning_rate = 0.001
epochs = 20
batch_size = 64
model = RoadSignClassifier.createCNN(width=30, height=30, depth=3, classes=43)
optimizer = Adam(lr=learning_rate, decay=learning_rate / (epochs))
```

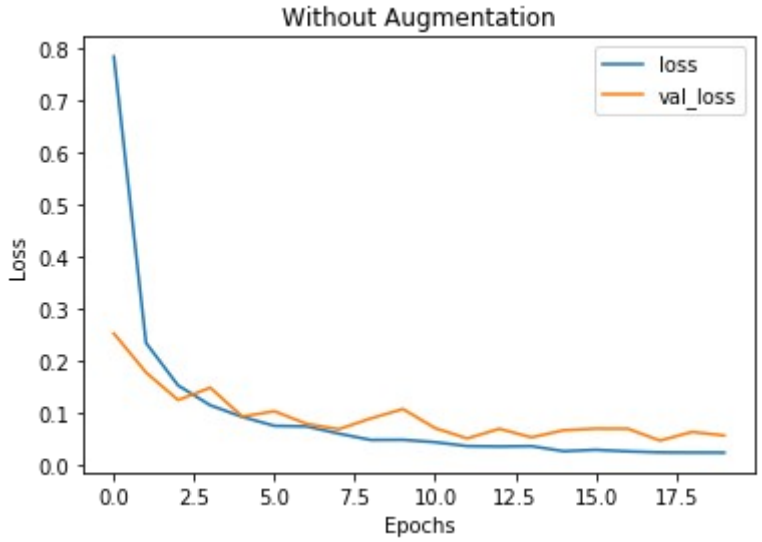
In [9]:
```
model.compile(optimizer=optimizer, loss="categorical_crossentropy", metrics=["accuracy"])
model_fit = model.fit(
    x=trainX,
    y=trainY,
    batch_size=batch_size,
    epochs=epochs,
    validation_split=0.2,
    class_weight=class_weight,
    verbose=1)
```

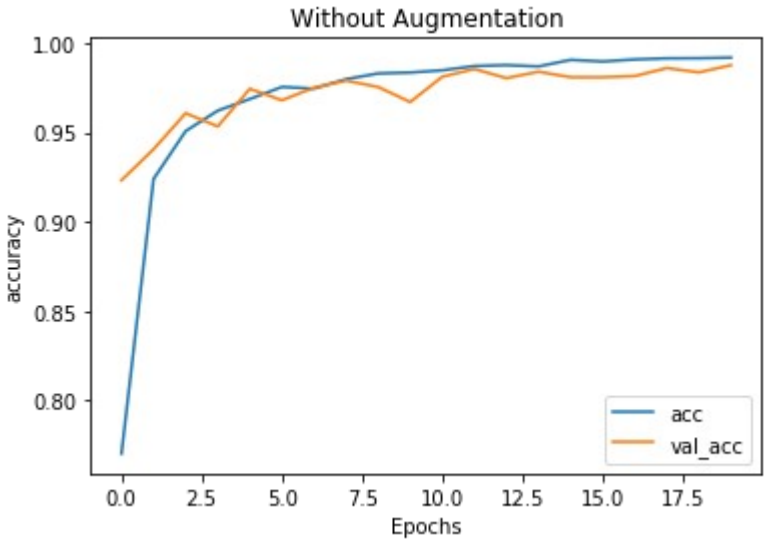
که در آن batch_size 64 برای ۲۰ epoch و با ۰.۰۰۱ learning rate که هر epoch از نسبت learning_rate/epoch کم‌تر می‌شود و از داده‌های نرمال شده استفاده کرده‌ایم و شبکه را آموزش می دهیم:

ب)بعد از اجرای شبکه فوق داریم:

```
Epoch 13/20
31367/31367 [==============================] - 21s 679us/sample - loss: 0.0370 - accuracy: 0.9876 - val_loss: 0.07
09 - val_accuracy: 0.9804
Epoch 14/20
31367/31367 [==============================] - 20s 652us/sample - loss: 0.0377 - accuracy: 0.9869 - val_loss: 0.05
45 - val_accuracy: 0.9839
Epoch 15/20
31367/31367 [==============================] - 24s 775us/sample - loss: 0.0280 - accuracy: 0.9906 - val_loss: 0.06
82 - val_accuracy: 0.9809
Epoch 16/20
31367/31367 [==============================] - 27s 876us/sample - loss: 0.0306 - accuracy: 0.9897 - val_loss: 0.07
13 - val_accuracy: 0.9809
Epoch 17/20
31367/31367 [==============================] - 25s 791us/sample - loss: 0.0276 - accuracy: 0.9908 - val_loss: 0.07
11 - val_accuracy: 0.9815
Epoch 18/20
31367/31367 [==============================] - 22s 689us/sample - loss: 0.0257 - accuracy: 0.9914 - val_loss: 0.04
83 - val_accuracy: 0.9860
Epoch 19/20
31367/31367 [==============================] - 23s 720us/sample - loss: 0.0254 - accuracy: 0.9915 - val_loss: 0.06
50 - val_accuracy: 0.9836
Epoch 20/20
31367/31367 [==============================] - 22s 704us/sample - loss: 0.0252 - accuracy: 0.9918 - val_loss: 0.05
81 - val_accuracy: 0.9876
```

که همان‌طور که دیده می‌شود به دقت تقریباً ۰.۹۹ رسیده‌ایم
نمودار های دقت و لاس به شکل زیر می شوند:





و سپس داده‌های تست را به شبکه می‌دهیم تا آن را ارزیابی کنیم و داریم:

In [10]:
```
test_loss, test_acc = model.evaluate(testX, testY, verbose=1)
```

```
12630/12630 [==============================] - 2s 133us/sample - loss: 0.3715 - accuracy: 0.9383
```

که دقت آن ۰.۹۳ و لاس آن ۰.۳۷ است

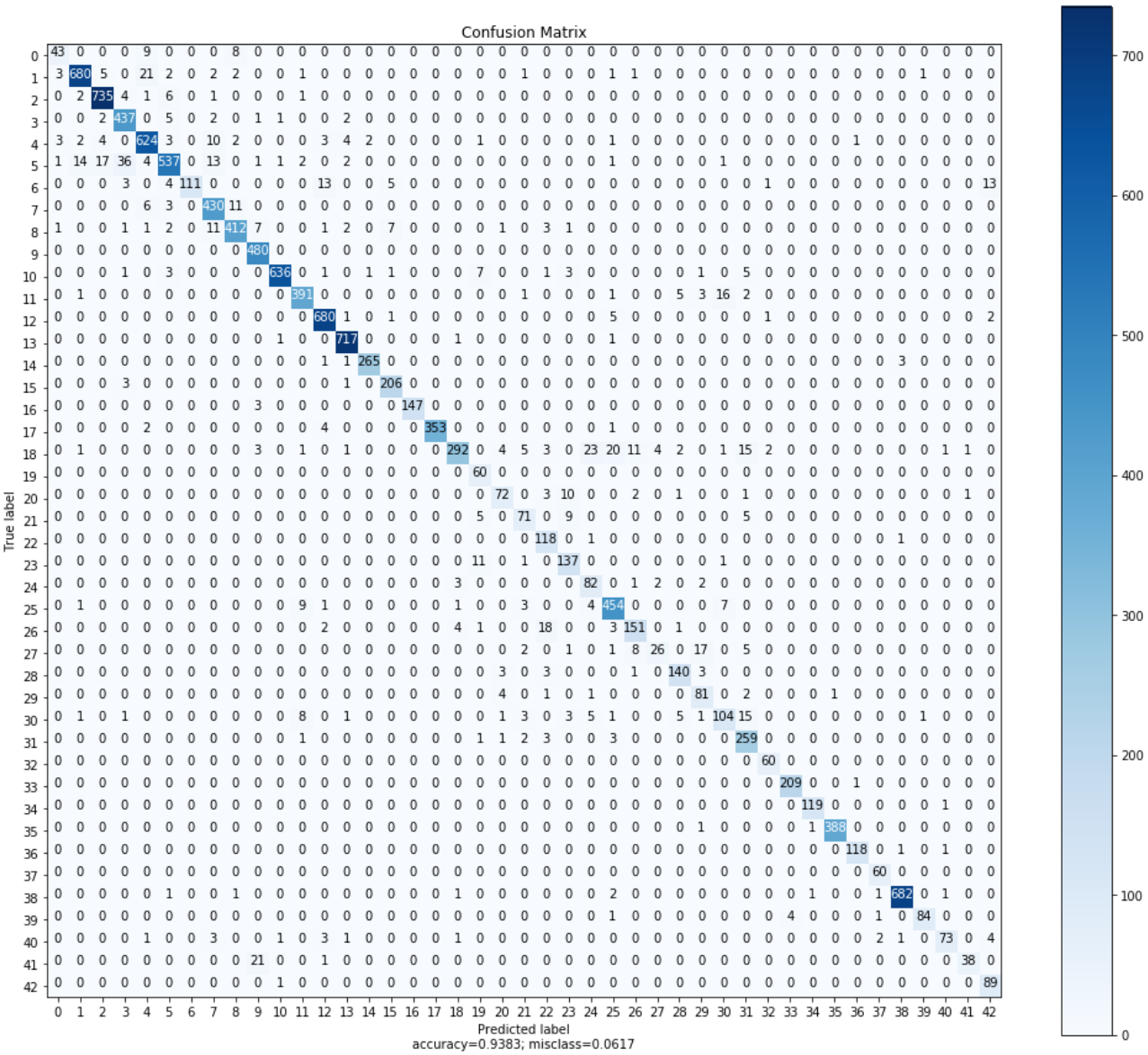
پ)

برای کشیدن ماتریس آشفتگی آن از کد زیر استفاده کرده ایم:

In [60]:
```
def plot_confusion_matrix(cm,
                          target_names = ['1', '2', '3', '4'],
                          title = 'Confusion matrix',
                          cmap = None,
                          normalize = False):

    import matplotlib.pyplot as plt
    import numpy as np
    import itertools

    accuracy = np.trace(cm) / float(np.sum(cm))
    misclass = 1 - accuracy

    if cmap is None:
        cmap = plt.get_cmap('Blues')

    plt.figure(figsize = (14, 12))
    plt.imshow(cm, interpolation = 'nearest', cmap = cmap)
    plt.title(title)
    plt.colorbar()

    if target_names is not None:
        tick_marks = np.arange(len(target_names))
        plt.xticks(tick_marks, target_names, rotation = 0)
        plt.yticks(tick_marks, target_names)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis = 1)[:, np.newaxis]


    thresh = cm.max() / 1.5 if normalize else cm.max() / 2
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        if normalize:
            plt.text(j, i, "{:0.4f}".format(cm[i, j]),
                     horizontalalignment = "center",
                     color = "white" if cm[i, j] > thresh else "black")
        else:
            plt.text(j, i, "{:,}".format(cm[i, j]),
                     horizontalalignment = "center",
                     color = "white" if cm[i, j] > thresh else "black")


    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label\naccuracy={:0.4f}; misclass={:0.4f}'.format(accuracy, misclass))
    plt.show()

```

و بعد از کشیدن آن داریم:

```
Matrix =
[[ 43   0   0 ...   0   0   0]
 [  3 680   5 ...   0   0   0]
 [  0   2 735 ...   0   0   0]
 ...
 [  0   0   0 ...  73   0   4]
 [  0   0   0 ...   0  38   0]
 [  0   0   0 ...   0   0  89]]
```



که همان‌طور که دیده می‌شود روی قطر اصلی بیشترین اعداد است که به معنی درست پیشبینی شدن آن‌ها ست و در سطر هایی اعداد دیگر نیز وجود دارند که به معنی غلط پیشبینی کردن است

ت)

شبکه فوق را به صورت زیر تغییر می‌دهیم تا برای ۳ تابع فعال ساز مختلف آن را تست کنیم

```
def createCustomActivationCNN(width, height, depth, classes, activation):
    model = Sequential()
    inputShape = (height, width, depth)
    model.add(Conv2D(filters=8, kernel_size=(5, 5), input_shape=inputShape, activation=activation))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Conv2D(filters=16, kernel_size=(3, 3), activation=activation))
    model.add(BatchNormalization())
    model.add(Conv2D(filters=16, kernel_size=(3, 3), activation=activation))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Conv2D(filters=32, kernel_size=(3, 3), padding="same", activation=activation))
    model.add(BatchNormalization())
    model.add(Conv2D(filters=32, kernel_size=(3, 3), padding="same", activation=activation))
    model.add(BatchNormalization())

    model.add(Flatten())
    model.add(Dropout(0.5))
    model.add(Dense(512, activation=activation))
    model.add(Dense(classes, activation="softmax"))
    return model
```

در این صورت تمام تابع های فعال ساز را با ۳ مورد relu sigmoid tanh امتحان می‌کنیم

شبکه را برای 15 epoch با batch size 64 و همان learning reate آموزش می دهیم:

```
In [65]:
learning_rate = 0.001
epochs = 15
batch_size = 64
# model = RoadSignClassifier.createCNN(width=30, height=30, depth=3, classes=43)
model_relu = RoadSignClassifier.createCustomActivationCNN(width=30, height=30, depth=3, classes=43,
                                                          activation="relu")
model_tanh = RoadSignClassifier.createCustomActivationCNN(width=30, height=30, depth=3, classes=43,
                                                          activation="tanh")
model_sigmoid = RoadSignClassifier.createCustomActivationCNN(width=30, height=30, depth=3, classes=43,
                                                             activation="sigmoid")
optimizer = Adam(lr=learning_rate, decay=learning_rate / (epochs))
```

بعد از اجرای هر کدام از آن‌ها داریم:

```
In [9]:
model_relu.compile(optimizer=optimizer, loss="categorical_crossentropy", metrics=["accuracy"])
model_relu_fit = model_relu.fit(
                x=trainX,
                y=trainY,
                batch_size=batch_size,
                epochs=epochs,
                validation_split=0.2,
                class_weight=class_weight,
                verbose=1)
```

```
Epoch 13/15
31367/31367 [==============================] - 18s 577us/sample - loss: 0.0345 - accuracy: 0.9880 - val_loss: 0.056
0 - val_accuracy: 0.9821
Epoch 14/15
31367/31367 [==============================] - 18s 561us/sample - loss: 0.0299 - accuracy: 0.9900 - val_loss: 0.052
4 - val_accuracy: 0.9861
Epoch 15/15
31367/31367 [==============================] - 18s 576us/sample - loss: 0.0295 - accuracy: 0.9894 - val_loss: 0.078
0 - val_accuracy: 0.9783
```

In [10]:
```
model_tanh.compile(optimizer=optimizer, loss="categorical_crossentropy", metrics=["accuracy"])
model_tanh_fit = model_tanh.fit(
                            x=trainX,
                            y=trainY,
                            batch_size=batch_size,
                            epochs=epochs,
                            validation_split=0.2,
                            class_weight=class_weight,
                            verbose=1)
```

```
Epoch 13/15
31367/31367 [==============================] - 18s 575us/sample - loss: 0.0276 - accuracy: 0.9916 - val_loss: 0.050
9 - val_accuracy: 0.9869
Epoch 14/15
31367/31367 [==============================] - 18s 571us/sample - loss: 0.0252 - accuracy: 0.9921 - val_loss: 0.051
2 - val_accuracy: 0.9861
Epoch 15/15
31367/31367 [==============================] - 18s 564us/sample - loss: 0.0245 - accuracy: 0.9921 - val_loss: 0.052
1 - val_accuracy: 0.9858
```

In [11]:
```
model_sigmoid.compile(optimizer=optimizer, loss="categorical_crossentropy", metrics=["accuracy"])
model_sigmoid_fit = model_sigmoid.fit(
                                    x=trainX,
                                    y=trainY,
                                    batch_size=batch_size,
                                    epochs=epochs,
                                    validation_split=0.2,
                                    class_weight=class_weight,
                                    verbose=1)
```

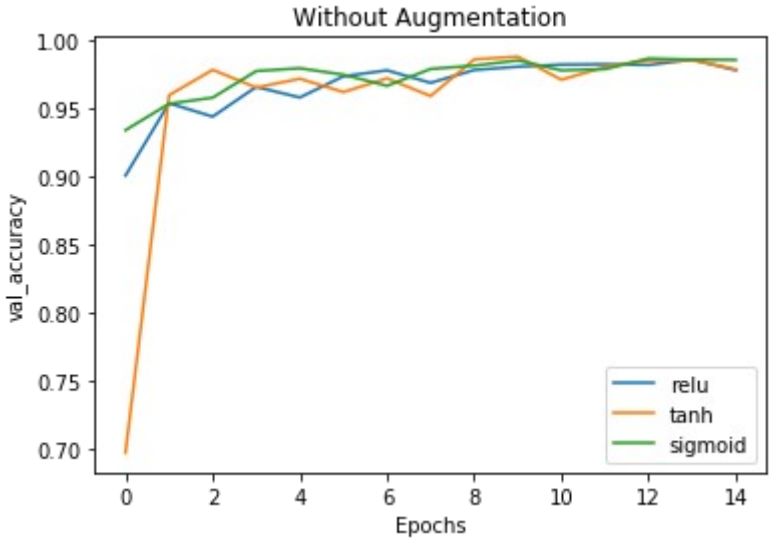
```
Train on 31367 samples, validate on 7842 samples
Epoch 13/15
31367/31367 [==============================] - 19s 614us/sample - loss: 0.0251 - accuracy: 0.9935 - val_loss: 0.052
8 - val_accuracy: 0.9855
Epoch 14/15
31367/31367 [==============================] - 23s 719us/sample - loss: 0.0209 - accuracy: 0.9948 - val_loss: 0.045
2 - val_accuracy: 0.9857
Epoch 15/15
31367/31367 [==============================] - 22s 711us/sample - loss: 0.0200 - accuracy: 0.9948 - val_loss: 0.069
1 - val_accuracy: 0.9790
```

و برای تست داریم:

In [13]:
```
test_loss_relu, test_acc_relu = model_relu.evaluate(testX, testY, verbose=1)
test_loss_tanh, test_acc_tanh = model_tanh.evaluate(testX, testY, verbose=1)
test_loss_sigmoid, test_acc_sigmoid = model_sigmoid.evaluate(testX, testY, verbose=1)
```

```
12630/12630 [==============================] - 2s 137us/sample - loss: 0.3284 - accuracy: 0.9320
12630/12630 [==============================] - 2s 155us/sample - loss: 0.3046 - accuracy: 0.9325
12630/12630 [==============================] - 2s 156us/sample - loss: 0.2537 - accuracy: 0.9384
```

و نمودار های آن به صورت زیر می شود:



همان‌طور که دیده می‌شود برای تست دقت هر ۳ تقریباً برابر بوده ولی مقدار لاس برای sigmoid و سپس relu , tanh بوده است ولی در نمودار هم دیده می‌شود که
برای validation accuracy دیده می‌شود که tanh بیشتر از دو تابع دیگر است
این به این خاطر است که تعداد لایه ها کانولوشن ای کم است و تابع relu , tanh فرق زیادی نمی‌کنند برای تعداد با لایه ی بالا تر relu جواب بهتری می‌دهید

ث)

حال می‌خواهیم همان مدل اول را که در زیر نیز آمده است یک بار با adam و یک بار با gradient decent آموزش دهیم و فرق این دو را ببینیم:

In [3]:
```
class RoadSignClassifier:
    def createCNN(width, height, depth, classes):
        model = Sequential()
        inputShape = (height, width, depth)
        model.add(Conv2D(filters=8, kernel_size=(5, 5), input_shape=inputShape, activation="relu"))
        model.add(MaxPooling2D(pool_size=(2, 2)))

        model.add(Conv2D(filters=16, kernel_size=(3, 3), activation="relu"))
        model.add(BatchNormalization())
        model.add(Conv2D(filters=16, kernel_size=(3, 3), activation="relu"))
        model.add(BatchNormalization())
        model.add(MaxPooling2D(pool_size=(2, 2)))

        model.add(Conv2D(filters=32, kernel_size=(3, 3), padding="same", activation="relu"))
        model.add(BatchNormalization())
        model.add(Conv2D(filters=32, kernel_size=(3, 3), padding="same", activation="relu"))
        model.add(BatchNormalization())

        model.add(Flatten())
        model.add(Dropout(0.5))
        model.add(Dense(512, activation="relu"))
        model.add(Dense(classes, activation="softmax"))
        return model
```

با هر دو تابع بهینه ساز SGD , ADAM شبکه را آموزش می‌دهیم و learning rate را همان ۰.۰۰۱ با کاهش learning rate/epoch قرار می‌دهیم
و داریم:

In [17]:
```
learning_rate = 0.001
epochs = 15
batch_size = 64
model_Adam = RoadSignClassifier.createCNN(width=30, height=30, depth=3, classes=43)
model_SGD = RoadSignClassifier.createCNN(width=30, height=30, depth=3, classes=43)
```

برای Adam داریم:

In [18]:
```
optimizer = Adam(lr=learning_rate, decay=learning_rate / (epochs))
model_Adam.compile(optimizer=optimizer, loss="categorical_crossentropy", metrics=["accuracy"])
model_Adam_fit = model_Adam.fit(
    x=trainX,
    y=trainY,
    batch_size=batch_size,
    epochs=epochs,
    validation_split=0.2,
    class_weight=class_weight,
    verbose=1)
```

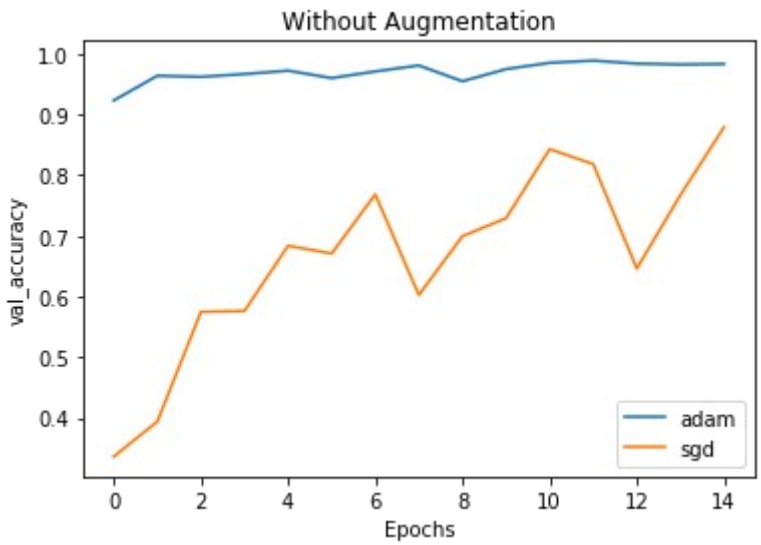
```
6 - val_accuracy: 0.9853
Epoch 12/15
31367/31367 [==============================] - 18s 586us/sample - loss: 0.0389 - accuracy: 0.9868 - val_loss: 0.044
9 - val_accuracy: 0.9890
Epoch 13/15
31367/31367 [==============================] - 18s 578us/sample - loss: 0.0281 - accuracy: 0.9904 - val_loss: 0.064
3 - val_accuracy: 0.9841
Epoch 14/15
31367/31367 [==============================] - 18s 581us/sample - loss: 0.0326 - accuracy: 0.9887 - val_loss: 0.066
2 - val_accuracy: 0.9827
Epoch 15/15
31367/31367 [==============================] - 18s 577us/sample - loss: 0.0319 - accuracy: 0.9893 - val_loss: 0.058
3 - val_accuracy: 0.9834
```

و برای SGD یا Gradient Decent داریم:

In [21]:
```
optimizer = SGD(lr=learning_rate, decay=learning_rate / (epochs))
model_SGD.compile(optimizer=optimizer, loss="categorical_crossentropy", metrics=["accuracy"])
model_SGD_fit = model_SGD.fit(
    x=trainX,
    y=trainY,
    batch_size=batch_size,
    epochs=epochs,
    validation_split=0.2,
    class_weight=class_weight,
    verbose=1)
```

```
Epoch 12/15
31367/31367 [==============================] - 17s 556us/sample - loss: 0.7967 - accuracy: 0.7766 - val_loss: 0.650
0 - val_accuracy: 0.8184
Epoch 13/15
31367/31367 [==============================] - 18s 570us/sample - loss: 0.7489 - accuracy: 0.7915 - val_loss: 1.372
3 - val_accuracy: 0.6463
Epoch 14/15
31367/31367 [==============================] - 19s 607us/sample - loss: 0.7080 - accuracy: 0.7997 - val_loss: 0.872
0 - val_accuracy: 0.7673
Epoch 15/15
31367/31367 [==============================] - 18s 590us/sample - loss: 0.6684 - accuracy: 0.8117 - val_loss: 0.481
4 - val_accuracy: 0.8791
```

و نمودار ان:



و برای تست آن داریم:

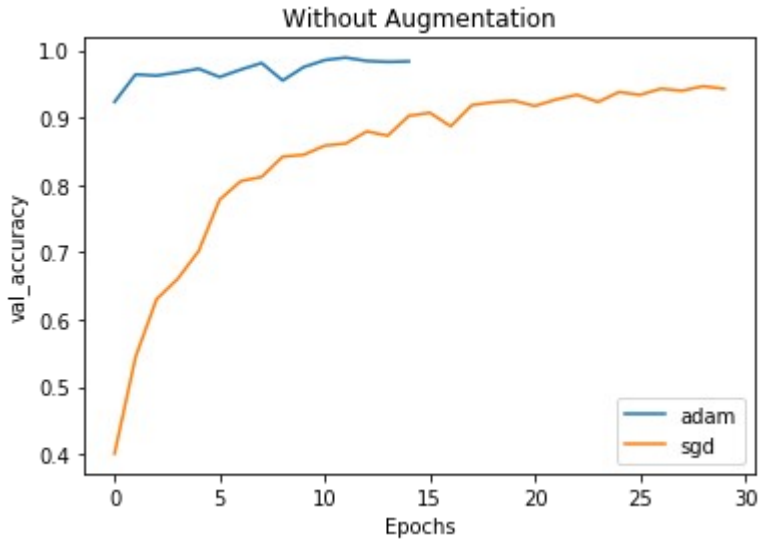
```
In [23]: 1 test_loss_adam, test_acc_adam = model_Adam.evaluate(testX, testY, verbose=1)
         2 test_loss_sgd, test_acc_sgd = model_SGD.evaluate(testX, testY, verbose=1)

12630/12630 [==============================] - 2s 138us/sample - loss: 0.3609 - accuracy: 0.9343
12630/12630 [==============================] - 2s 147us/sample - loss: 0.6648 - accuracy: 0.8253
```

می‌بینیم که sgd جواب خیلی بد تری به ما می‌دهد زیرا می‌دانیم adam نیز با استفاده از همان الگوریتم و بهبود یافته ی آن بهینه سازی می‌کند و دقت بالا تری می‌گیرد ولی sgd الگوریتم ضعیفی است و نیاز به تعداد epoch بیشتری برای آموزش دارد تا بتواند به دقت قابل قبولی برسد در حالی که adam می‌تواند این کار را سریع‌تر انجام دهد

برای SGD اگر تعداد epoch را بیشتر کنیم دقت بالاتری می‌توانیم بگیریم مثلاً برای ۳۰ epoch داریم:

```
Epoch 27/30
31367/31367 [==============================] - 18s 569us/sample - loss: 0.3166 - accuracy: 0.9133 - val_loss: 0.212
1 - val_accuracy: 0.9429
Epoch 28/30
31367/31367 [==============================] - 18s 582us/sample - loss: 0.3071 - accuracy: 0.9140 - val_loss: 0.215
1 - val_accuracy: 0.9398
Epoch 29/30
31367/31367 [==============================] - 18s 570us/sample - loss: 0.2950 - accuracy: 0.9184 - val_loss: 0.196
4 - val_accuracy: 0.9466
Epoch 30/30
31367/31367 [==============================] - 20s 647us/sample - loss: 0.2902 - accuracy: 0.9202 - val_loss: 0.199
6 - val_accuracy: 0.9427
```

که می‌بینیم هنوز هم در حال زیاد شدن دقت است پس اگر باز هم زیاد کنیم بیشتر می‌شود ولی ماکسیموم مقدار آن از adam همیشه کمتر است زیرا adam بیهنه تر است نمودار آن به صورت زیر می شود:



که بعد از ۳۰ epoch هنوز به دقت adam نرسیده است

ج)

حال میخواهیم شبکه را برای حالت با drop out و بدون آن امتحان کنیم می‌دانیم drop out برای لایه‌های fully connected است که به میزان درصدی از نورون ها را حذف می‌کند تا از overfit شدن جلوگیری شود

برای شبکه همان شبکه قبلی یک بار با dropout و یک بار بدون drop out داریم: برای ۱۰۰ epoch شبکه را آموزش می دهیم

In [ ]:
```
learning_rate = 0.001
epochs = 100
batch_size = 64
model_with_drop_out = RoadSignClassifier.createCNN(width=30, height=30, depth=3, classes=43)
model_without_drop_out = RoadSignClassifier.createCNN_without_dropout(width=30, height=30, depth=3, classes=43)

```

سپس برای حالت با dropout داریم:

In [29]:
```
model_with_drop_out.compile(optimizer=optimizer, loss="categorical_crossentropy", metrics=["accuracy"])
model_with_drop_out_fit = model_with_drop_out.fit(
    x=trainX,
    y=trainY,
    batch_size=batch_size,
    epochs=epochs,
    validation_split=0.2,
    class_weight=class_weight,
    verbose=1)
```

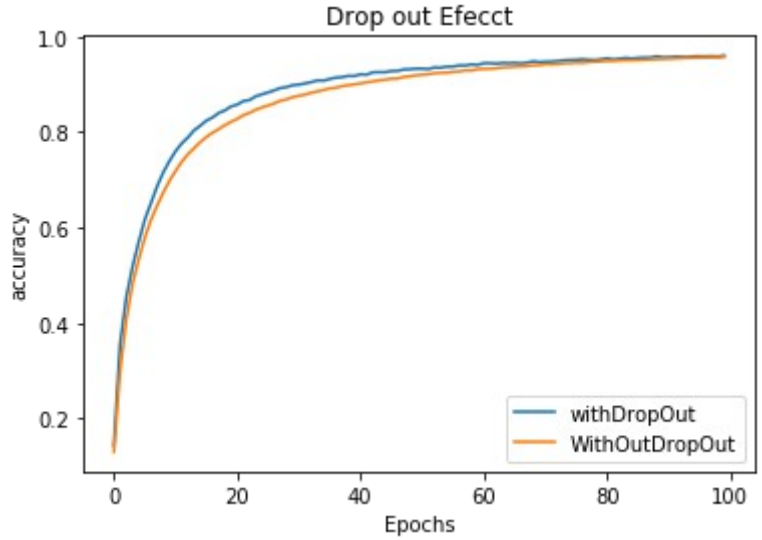
```
31367/31367 [==============================] - 17s 537us/sample - loss: 0.1555 - accuracy: 0.9571 - val_loss: 0.1
057 - val_accuracy: 0.9727
Epoch 95/100
31367/31367 [==============================] - 17s 543us/sample - loss: 0.1532 - accuracy: 0.9585 - val_loss: 0.1
027 - val_accuracy: 0.9712
Epoch 96/100
31367/31367 [==============================] - 17s 538us/sample - loss: 0.1517 - accuracy: 0.9588 - val_loss: 0.0
973 - val_accuracy: 0.9740
Epoch 97/100
31367/31367 [==============================] - 17s 542us/sample - loss: 0.1511 - accuracy: 0.9586 - val_loss: 0.0
963 - val_accuracy: 0.9732
Epoch 98/100
31367/31367 [==============================] - 17s 533us/sample - loss: 0.1490 - accuracy: 0.9590 - val_loss: 0.0
976 - val_accuracy: 0.9732
Epoch 99/100
31367/31367 [==============================] - 17s 537us/sample - loss: 0.1488 - accuracy: 0.9584 - val_loss: 0.0
960 - val_accuracy: 0.9737
Epoch 100/100
31367/31367 [==============================] - 17s 541us/sample - loss: 0.1445 - accuracy: 0.9602 - val_loss: 0.0
944 - val_accuracy: 0.9733
```

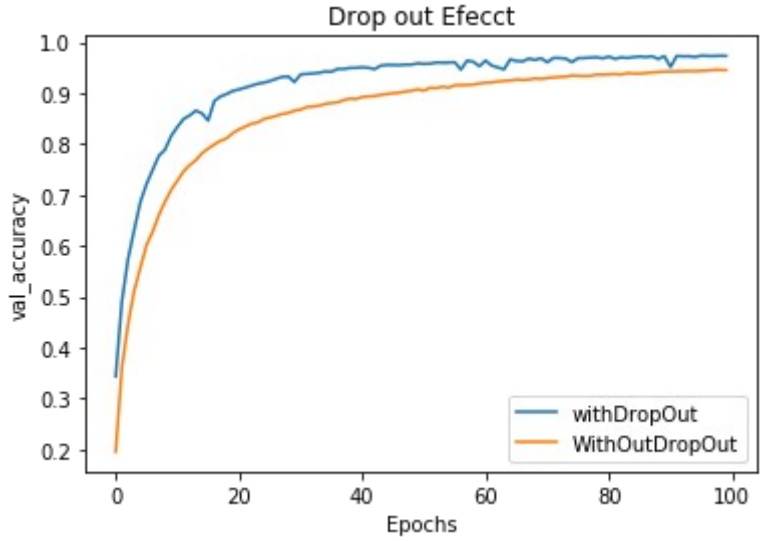
و برای حالت بدون آن داریم:

In [30]:
```
model_without_drop_out.compile(optimizer=optimizer, loss="categorical_crossentropy", metrics=["accuracy"])
model_without_drop_out_fit = model_without_drop_out.fit(
    x=trainX,
    y=trainY,
    batch_size=batch_size,
    epochs=epochs,
    validation_split=0.2,
    class_weight=class_weight,
    verbose=1)
```

```
31367/31367 [==============================] - 16s 523us/sample - loss: 0.1990 - accuracy: 0.9553 - val_loss: 0.2
278 - val_accuracy: 0.9436
Epoch 95/100
31367/31367 [==============================] - 17s 529us/sample - loss: 0.1969 - accuracy: 0.9557 - val_loss: 0.2
265 - val_accuracy: 0.9434
Epoch 96/100
31367/31367 [==============================] - 16s 513us/sample - loss: 0.1952 - accuracy: 0.9560 - val_loss: 0.2
239 - val_accuracy: 0.9443
Epoch 97/100
31367/31367 [==============================] - 17s 538us/sample - loss: 0.1939 - accuracy: 0.9565 - val_loss: 0.2
227 - val_accuracy: 0.9445
Epoch 98/100
31367/31367 [==============================] - 16s 518us/sample - loss: 0.1920 - accuracy: 0.9564 - val_loss: 0.2
216 - val_accuracy: 0.9457
Epoch 99/100
31367/31367 [==============================] - 17s 536us/sample - loss: 0.1910 - accuracy: 0.9567 - val_loss: 0.2
197 - val_accuracy: 0.9458
Epoch 100/100
31367/31367 [==============================] - 17s 531us/sample - loss: 0.1892 - accuracy: 0.9576 - val_loss: 0.2
185 - val_accuracy: 0.9454
```

برای نمودار های آن داریم:





و برای تست آن داریم:

```
In [33]: 1 test_loss_with_drop_out, test_acc_with_drop_out = model_with_drop_out.evaluate(testX, testY, verbose=1)
         2 test_loss_wihtout_drop_out, test_acc_wihtout_drop_out = model_without_drop_out.evaluate(testX, testY, verbose=1)

         12630/12630 [==============================] - 8s 626us/sample - loss: 0.2366 - accuracy: 0.9317
         12630/12630 [==============================] - 2s 119us/sample - loss: 0.4034 - accuracy: 0.8864
```

همان‌طور که دیده می‌شود دقت شبکه بدون dropout پایین‌تر و لاس آن بالاتر است برای داده‌های تست این نسبت بیشتر هم هست زیرا شبکه با drop out می‌تواند از حفظ کردن جلوگیری کند و با دقت بهتری یاد بگیرد
شبکه ممکن بود بدون drop out over fit نیز شود البته در اینجا اتفاق نیافتاده است

ج)

در اینجا می‌خواهیم تأثیر Augment کردن داده‌ها را برای آموزش شبکه ببینیم برای همین یک بار شبکه را با داده‌های معمولی و یک بار با اگمنت شده های آن برای ۱۰۰ epoch آموزش می‌دهیم
برای augment کردن داده‌ها مانند زیر عمل می کنیم:

In [7]:
```
data_aug = ImageDataGenerator(
rotation_range=10,
zoom_range=0.15,
width_shift_range=0.1,
height_shift_range=0.1,
shear_range=0.15,
horizontal_flip=False,
vertical_flip=False)
```

که همان‌طور که دیده می‌شود آن‌ها را چرخانده زوم کرده از عرض و طول کشیده می‌کنیم تا داده‌های جدید را بدست اوریم
در اینجا آن‌ها را flip نکرده‌ایم زیرا ممنک است معنی تابلو ها نیز عوض شوند
۲ مدل برای آن‌ها درست می کنیم: و آن‌ها را با batch size = 64 انجام می دهیم:

In [ ]:
```
learning_rate = 0.001
epochs = 100
batch_size = 64
model_without_augment = RoadSignClassifier.createCNN(width=30, height=30, depth=3, classes=43)
model_with_augment = RoadSignClassifier.createCNN(width=30, height=30, depth=3, classes=43)

```

و آن‌ها را آموزش می‌دهیم و داریم:
برای بدون augment داریم:

In [9]:
```
model_without_augment.compile(optimizer=optimizer, loss="categorical_crossentropy", metrics=["accuracy"])
model_without_augment_fit = model_without_augment.fit(
    x=trainX,
    y=trainY,
    batch_size=batch_size,
    epochs=epochs,
    validation_split=0.2,
    class_weight=class_weight,
    verbose=1)
```

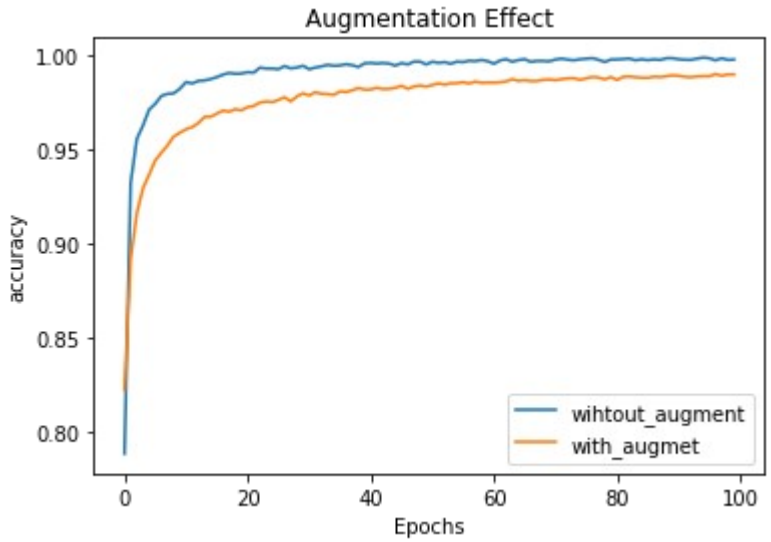
```
31367/31367 [==============================] - 23s 731us/sample - loss: 0.0051 - accuracy: 0.9985 - val_loss: 0.0
374 - val_accuracy: 0.9940
Epoch 95/100
31367/31367 [==============================] - 23s 738us/sample - loss: 0.0027 - accuracy: 0.9991 - val_loss: 0.0
465 - val_accuracy: 0.9920
Epoch 96/100
31367/31367 [==============================] - 22s 712us/sample - loss: 0.0041 - accuracy: 0.9987 - val_loss: 0.0
433 - val_accuracy: 0.9927
Epoch 97/100
31367/31367 [==============================] - 23s 720us/sample - loss: 0.0073 - accuracy: 0.9976 - val_loss: 0.0
423 - val_accuracy: 0.9918
Epoch 98/100
31367/31367 [==============================] - 22s 715us/sample - loss: 0.0043 - accuracy: 0.9987 - val_loss: 0.0
475 - val_accuracy: 0.9921
Epoch 99/100
31367/31367 [==============================] - 23s 738us/sample - loss: 0.0065 - accuracy: 0.9979 - val_loss: 0.0
423 - val_accuracy: 0.9921
Epoch 100/100
31367/31367 [==============================] - 23s 729us/sample - loss: 0.0061 - accuracy: 0.9980 - val_loss: 0.0
391 - val_accuracy: 0.9927
```

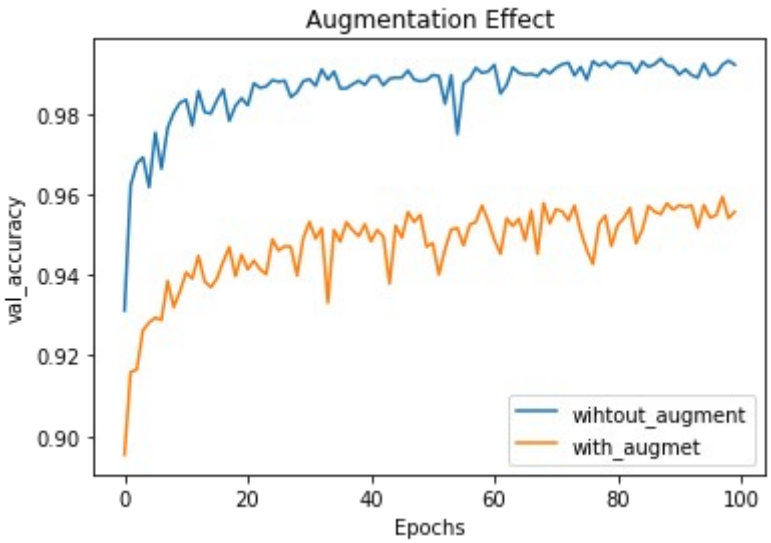
و برای Augmented data داریم:

```
In [10]:
model_with_augment.compile(optimizer=optimizer, loss="categorical_crossentropy", metrics=["accuracy"])
model_with_augment_fit = model_without_augment.fit(
    data_aug.flow(trainX, trainY, batch_size=batch_size),
    epochs=epochs,
    validation_data=(testX, testY),
    class_weight=class_weight,
    verbose=1)
```

```
613/613 [==============================] - 27s 44ms/step - loss: 0.0335 - accuracy: 0.9890 - val_loss: 0.1044 - v
al_accuracy: 0.9740
Epoch 95/100
613/613 [==============================] - 27s 44ms/step - loss: 0.0337 - accuracy: 0.9891 - val_loss: 0.1070 - v
al_accuracy: 0.9746
Epoch 96/100
613/613 [==============================] - 27s 44ms/step - loss: 0.0315 - accuracy: 0.9892 - val_loss: 0.1055 - v
al_accuracy: 0.9755
Epoch 97/100
613/613 [==============================] - 27s 44ms/step - loss: 0.0293 - accuracy: 0.9903 - val_loss: 0.1275 - v
al_accuracy: 0.9712
Epoch 98/100
613/613 [==============================] - 27s 44ms/step - loss: 0.0319 - accuracy: 0.9894 - val_loss: 0.1175 - v
al_accuracy: 0.9727
Epoch 99/100
613/613 [==============================] - 27s 44ms/step - loss: 0.0323 - accuracy: 0.9901 - val_loss: 0.1108 - v
al_accuracy: 0.9725
Epoch 100/100
613/613 [==============================] - 27s 44ms/step - loss: 0.0312 - accuracy: 0.9901 - val_loss: 0.1034 - v
al_accuracy: 0.9755
```

و نمودار ها به صورت:





و برای داده‌های تست داریم:

```
test_loss_without_augment, test_acc_without_augment = model_without_augment.evaluate(testX, testY, verbose=1)
test_loss_with_augment, test_acc_with_augment = model_with_augment.evaluate(testX, testY, verbose=1)
```

```
12630/12630 [==============================] - 2s 133us/sample - loss: 0.1034 - accuracy: 0.9755
12630/12630 [==============================] - 2s 145us/sample - loss: 3.7596 - accuracy: 0.0769
```