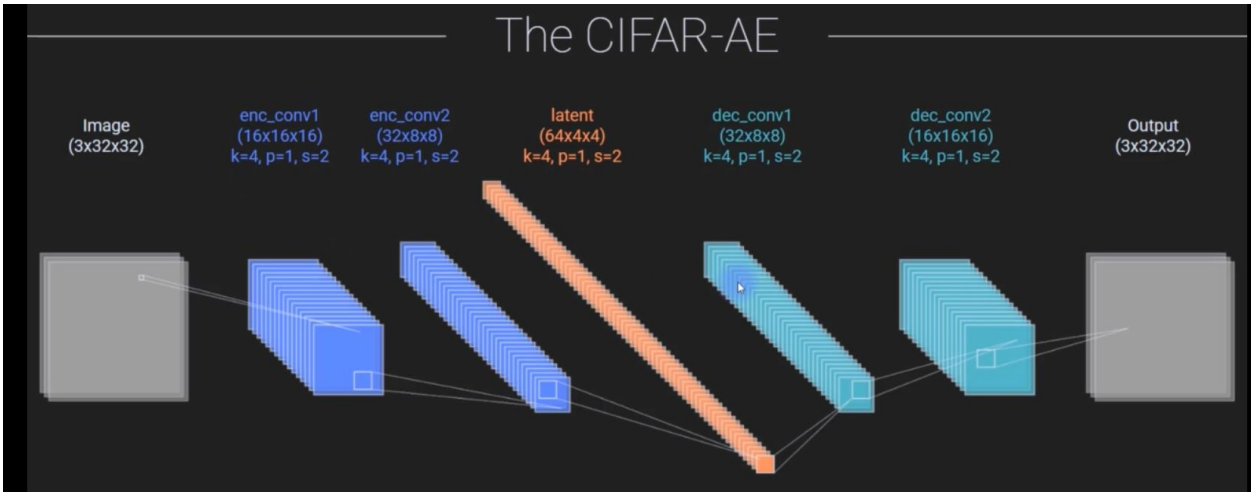
Project 2: CIFAR10 Autoencoders

با توجه به داده های CIFAR10 و توجه به الگوی زیر اتوانکودر و دی کودر ها با مشخصات داده شده را بنویسید.



استفاده از اتوانکودر باعث بهتر شدن شرایط تصویر از جمله حذف بعضی نویزهای روی تصویر و مشخص تر شدن سوژه تصویر می گردد.

گام های برنامه و نکات آن:

CIFAR10_Encoder_01.ipynb **OR** Rewrite_CIFAR10_Autoencoders.ipynb

- وارد کردن کتابخانه ها و معرفی GPU
- خواندن داده ها : در خواندن داده ها از توابع تبدیل استفاده می کنیم که در زمان استفاده از دیتالودر کار تبدیل داده ها (تبدیل به تنسور و نرمالایز کردن) اتفاق افتد.
- ایجاد دیتالودر و مشخص کردن بچ سایز
- ایجاد مدل اتو انکودر شامل انکودر و دی کودر مشابه تصویر فوق :
  - در ایجاد این مدل از مکس پولینگ استفاده نمی کنیم و با استفاده از استراید اندازه ماتریس ها (تصاویر) در لایه ها بعدی ایجاد می شود
  - پس از هر کانولوشن یک تابع فعال ساز لیکی رلو استفاده می شود.
  - سه لایه انکودر و سپس سه لایه دی کودر در مدل در قسمت init تعریف می شود و لیکی رلو ها و معرفی داده های ورودی (X) در قسمت تابع فوروارد انجام می شود.

- از بهینه ساز آدام استفاده می کنیم و نرخ یادگیری ۰,۰۰۱ و weight_decay = 0.00001 انتخاب می کنیم
- تایع خطا برای این مساله میانگین مربع خطا (MSE) می باشد.

- پس از تعریف مدل شبکه عصبی تابعی جهت یادگیری مدل تعریف می کنیم که ورودی های آن شبکه عصبی تعریف شده، تابع خطا و نیز تابع بهینه ساز می باشد. در واقع این سه مورد خروجی تابع ساخت شبکه عصبی می باشند و بعنوان ورودی به این تابع در نظر گرفته می شوند.

```
def funtion2trainTheAEModel(net,lossfun,optimizer):
```

- در این تابع مشخص می کنیم که چند بار تابع مسیر یادگیری (epoch) را تکرار کند.
- در ابتدای این تابع شبکه عصبی را به GPU معرفی می کنیم.
- در هر مرحله (epoch) مقدار خطای آموزش و خطای تست را ذخیره می کنیم تا در نمودار ترسیم نماییم.
- برای ایجاد حلقه به تعداد epoch از یک حلقه for استفاده می کنیم.
  - در داخل حلقه بالا دو حلقه دیگر که یکی برای خواندن داده های آموزش و پس از آن حلقه دوم برای خواندن داده های تست می باشد قرار می دهیم. در واقع پس از هر بار آموزش یک بار مدل را تست می کنیم تا عملکرد آن را بسنجیم. پس از یک بار طی مسیر آموزش و تست وارد مرحله (epoch) دوم یعنی حلقه بالایی می شویم.
  - قبل از ورود به حلقه های آموزش و تست بهتر است به برنامه یادآوری کنیم که وارد فاز آموزش (net.train()) و فاز ارزیابی (net.eval()) می گردد.
  - در داخل این حلقه ها باید داده های آموزش و تست را از دیتالودر به صورت بچ به بچ خوانده و به GPU معرفی کنیم.
  - برای هر بچ داده با استفاده از شبکه معرفی شده در ورودی تابع (net) مقدار لیبل را پیش بینی می کنیم (yhat)
  - همچنین برای هر بچ داده مقدار خطا را نیز با استفاده از تابع خطای معرفی شده در ورودی ارزیابی می نماییم. (loss)
  - **نکته** اینکه با توجه به اینکه در اینجا اتوانکودر در شبکه آموزش می بیند، مقایسه باید با تصویر ورودی صورت پذیرد. در واقع می خواهیم مقداری نویز و پیکسل های اضافی تصویر ورودی را حذف کنیم و هدف (target) بدست آمده با تصویر ورودی مقایسه می شود.
  - سپس مسیر برگشت (Back propagation) را تعریف کرده و پس از آن مقدار خطای بچ بارگذاری شده در شبکه را ذخیره می کنیم.
  - باید دقت کنیم که مقدار خطا را باید با استفاده از تابع item() بصورت عددی ذخیره نماییم.
    - batchloss.append(loss.item())
- سپس در حلقه مراحل یادگیری (epochs) مقدار خطای یک مرحله کامل (one epoch) را از میانگین خطای بچ های داده وارد شده به مدل محاسبه و ذخیره می کنیم.

- در مرحله بعد باید وارد فاز ارزیابی مدل گردیم. در اینجا به برنامه یادآوری می کنیم که شبکه وارد فاز ارزیابی می گردد (net.eval())
  - سپس حلقه دوم یعنی وارد کردن داده های تست به شبکه صورت می گیرد و داده ها بصورت بچ به بچ با بچ سایز داده شده برای داده های تست به شبکه وارد و پیش بینی برای لیبل (yHat) انجام می شود.
  - مجددا مشابه حلقه آموزشی، خطای هر بچ داده با استفاده از تابع خطای معرفی شده به تابع آموزش مدل ارزیابی و برای هر بچ ذخیره می گردد.
  - **نکته مهم** اینکه در ارزیابی مدل و استفاده از داده های تست دیگر مسیر برگشت (Back Propagation) وجود ندارد و ارزیابی در انتهای مسیر رفت صورت می گیرد. لذا برای اینکه در هر مسیر مطمئن شویم که مقدار خطا ها از قبل در حافظه باقی نمانده است از دستورات زیر استفاده می کنیم:

```
with torch.no_grad():
    yHat = net(X)
    loss = lossfun(yHat, X)
```

  - در واقع در انتهای دستور with شبکه به کار خود پایان می دهد و پارامتر های آن از حافظه پاک می شوند.
- پس از وارد شدن کلیه داده های تست به شبکه و ارزیابی خطای آن بصورت یچ به بچ، با استفاده از میانگین آن خطاها، خطای مرحله تست (epoch) را ذخیره می کنیم.
- در اینجا یک حلقه کامل آموزش و تست و ذخیره سازی خطاهای آنها کامل می گردد و تا کامل شدن تعدادی که برای این تکرار مشخص کرده ایم ادامه می یابد.
- خروجی این تابع را میزان خطای آموزش، میزان خطای تست و شبکه معرفی شده (ورودی تابع) قرار می دهیم.

```
return trainLoss,testLoss,net
```

- با استفاده از توابع تعریف شده فوق کار آموزش و ارزیابی مدل با دو خط برنامه و فراخوانی توابع انجام می شود.

```
netAE, lossfun, optimizer = makeTheAENet()

trainLossAE, testLossAE, netAE = funtion2trainTheAEModel(netAE, lossfun,
optimizer)
```

- سپس ماتریس های خطای آموزش و خطای تست را در نموداری ترسیم می کنیم.