

## گزارش تمرین سوم شبکه عصبی

علیرضا آزادبخت ۹۹۴۲۲۰۱۹

### تمرین ۱:

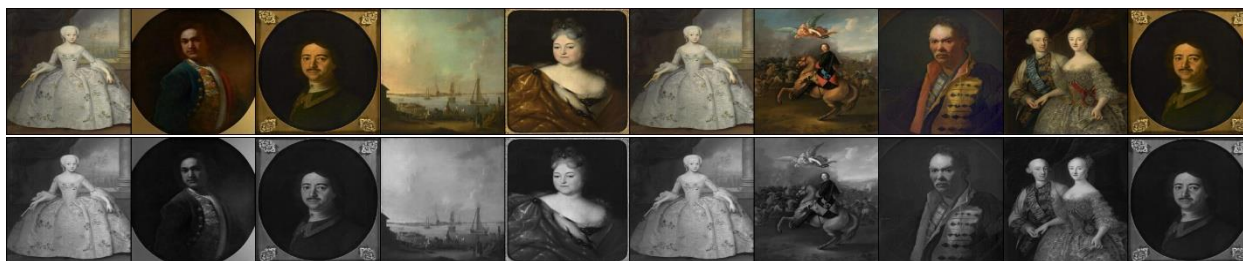
#### مسئله رنگ آمیزی عکس های سیاه و سفید:

این مسئله شامل ۲۰۲۴ تابلوی نقاشی قدیمی از هنرمندان نسبتاً مطرح در طول تاریخ می باشد که در ابعاد مختلف در اختیار ما قرار گرفته هدف ما طراحی معماری های مختلف اوتوانکدر ها و بررسی هایپرپارامتر های مختلف آن هاست که بتوانند به کمک این دیتای ست عکس های سیاه و سفید را رنگ آمیزی کنند، رویکرد رو به جلو قابل استفاده ابتدا سیاه و سفید کردن نقاشی ها به عنوان ورودی شبکه و قرار دادن عکس اصلی به عنوان هدف خروجی شبکه میباشد.



#### سوال اول: سیاه و سفید کردن عکس ها؟

همانطور که گفتیم برای ساختن دیتاست مورد نیاز ابتدا عکس های نقاشی ها را به ابعاد ۲۵۶ در ۲۵۶ اسکیل کردیم که تمامی نقاشی ها دارای ابعاد یکسانی باشند، سپس به کمک تابع `rgb2gray` کتابخانه `skimage` عکس ها را سیاه و سفید کردیم و سپس دیتا لودری مناسب پارتورچ از آن ها با `batch size = 32` تهیه کردیم که در آن ها بردار ورودی  $X$  عکس سیاه سفید با یک چنل رنگی بود و بردار خروجی و هدف  $y$  عکس رنگی همان تابلو قرار داشت.



## سوال دوم: تابع خطا مناسب چیست؟

بعد از مطالعه کد ها و مقالات انجام شده در این زمینه متوجه شدیم که تابع هزینه متداول برای این تسک تابع MSE Loss می باشد، اما این تابع مشکلاتی دارد: چون مسئله رنگ آمیزی عکس ها یک مسئله چند جوابه است استفاده از این تابع هزینه باعث میشود که مدل ها به سمت پیشبینی رنگ های روشن و شفاف نروند و خروجی های معمولا رنگی بین سه کانال رنگی قرمز و سبز و آبی باشد و معمولا رنگی معادل قهوه ای خروجی می دهند، و در دیتا ست فعلی ما اکثر نقاشی ها تقریبا همین رنگ را نیز دارا هستند پس مقدار خطا با انتخاب همین طیف رنگی مقدار مناسبی می شود، اما در دیتا ست های متنوع تر تابع هزینه متفاوتی استفاده میشود و بهترین مدل فعلی برای این تسک یک مرحله کلسیفیکیشن انجام میدهد و یک مرحله رگرسیون مقدار رنگ پیکسل ها، بدین صورت که گفتیم مسئله رنگ آمیزی لزوما یک جواب ندارد و یک پیکسل سیاه و سفید می تواند به هر سه مقدار آبی و قرمز و سبز متمایل باشد برای همین در کار های جدید تر ابتدا مدل طبقه بندی میکند که کدام رنگ در چنل های خروجی مقدار بیشتری را دارد و یک مرحله طبقه بندی ۳ کلاسه بین رنگ های آبی و قرمز و سبز انجام می دهد سپس با ضریبی معادل پیشبینی احتمالاتی آن کلاس اقدام به رنگ آمیزی پیکسل می کند به کمک این روش مشکل رنگ آمیزی های قهوه ای طور حل شده و خروجی های شفاف و رنگی تری بدست می آید.

## سوال سه: ساخت دیتا ست.

همانطور که در بخش اول گفته شد دیتا ست و سپس دیتا لودری از روی آن جهت کارایی بهینه و بهتر از مجموعه دادگان ساخته شد.

## سوال چهار: آموزش شبکه های اتو انکدری و بررسی پارامتر های آن:

به طور کلی چهار معماری مختلف از اتو انکدر ها تهیه کردیم، یک معماری کوچک و غیر عمیق و بر روی آن تاثیر مقادیر مختلف نرخ یادگیری الگوریتم های بهینه سازی مختلف را بررسی کردیم، سپس یک معماری عمیق

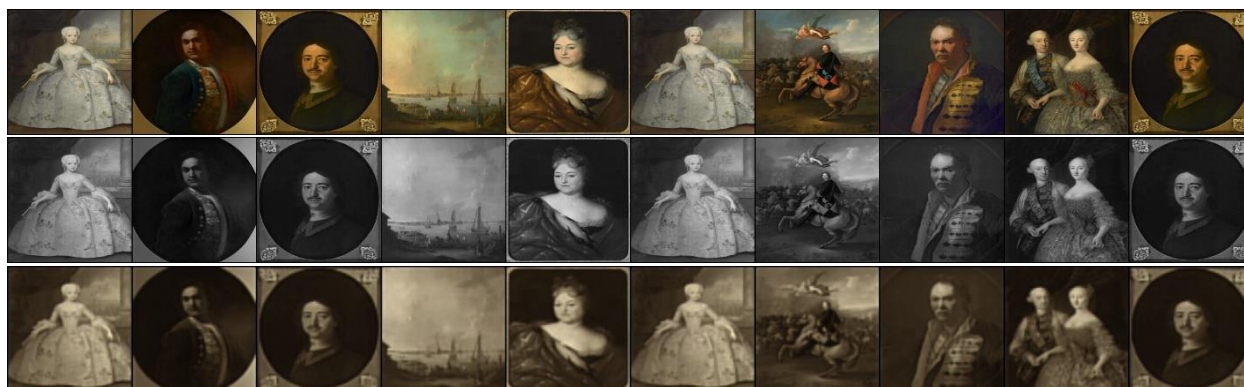
و یک معماری به کمک شبکه از پیش آموزش دیده inception\_v3 تهیه کردیم که انکدر آن inception\_v3 بود و به جای لایه آخر آن یک دیکدر متداول بر پایه معماری کوچک قرار دادیم و نتایج را بررسی کردیم، در مرحله بعد skip connection ها را اضافه کردیم و یک مدل بر پایه معماری U-net ساختیم.

### معماری کوچک:

این معماری شامل سه لایه شبکه کانولوژنی با کانال های ۳۲ و ۳۲ و ۶۴ در قسمت انکدر است و شامل ۳ لایه کانولوژنی وارون (transposed conv) است که عکس با یک چنل را دریافت و عکس رنگی با ۳ چنل خروجی می دهد

آزمایش یک نرخ یادگیری:

این شبکه با نرخ یادگیری ۰.۰۰۰۱ و بهینه ساز adam و ۵ اپاک به کمک کتابخانه های torchbearer آموزش داده شد و به نتایج زیر رسیدیم.



فرایند یادگیری	خطا بر روی مجموعه داده تست	خطا بر روی مجموعه داده آموزش
	0.0038	0.0043

آزمایش دو نرخ یادگیری:

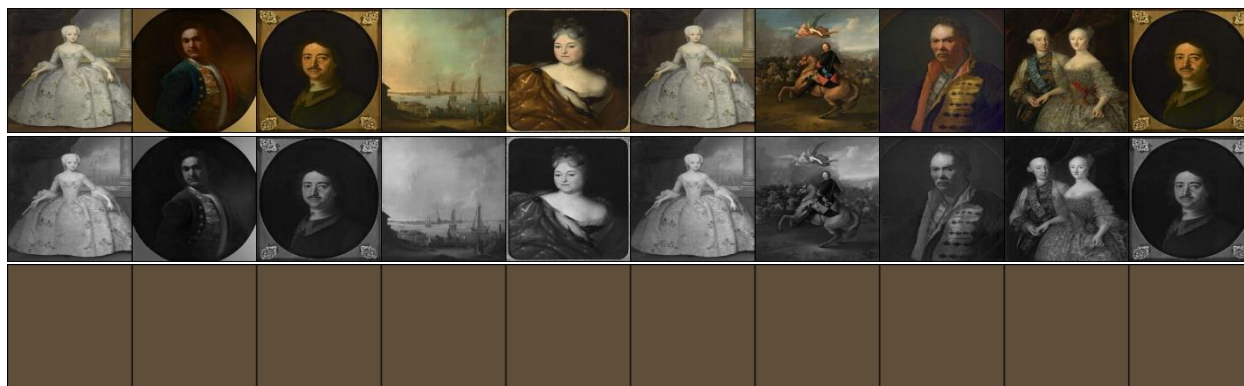
این شبکه با نرخ یادگیری ۰.۰۰۱ و بهینه ساز adam و ۵ اپاک به کمک کتابخانه های torchbearer آموزش داده شد و به نتایج زیر رسیدیم.

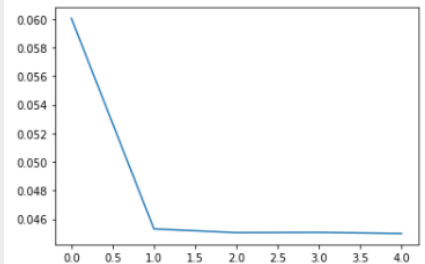


فرایند یادگیری	خطا بر روی مجموعه داده تست	خطا بر روی مجموعه داده آموزش
	0.0028	0.0035

آزمایش سه نرخ یادگیری:

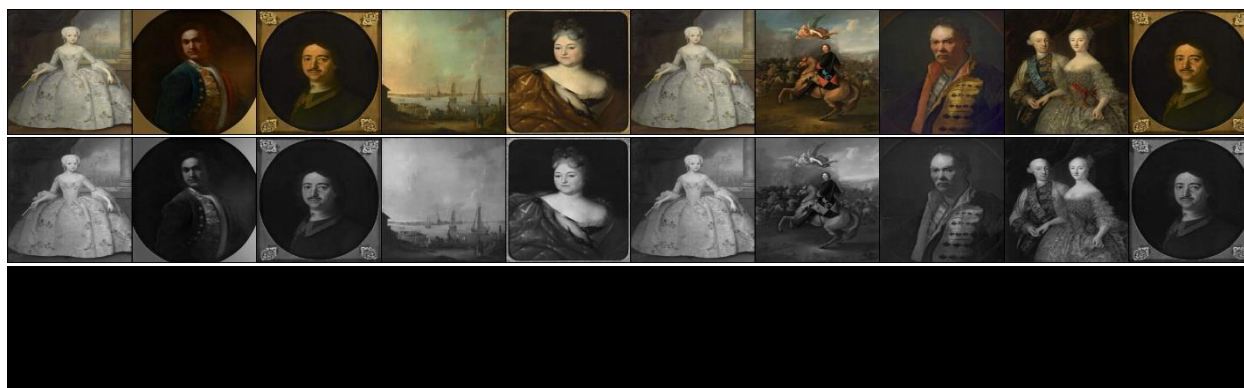
این شبکه با نرخ یادگیری ۰.۰۰۱ و بهینه ساز adam و ۵ اپاک به کمک کتابخانه های torchbearer آموزش داده شد و به نتایج زیر رسیدیم.



فرایند یادگیری	خطا بر روی مجموعه داده تست	خطا بر روی مجموعه داده آموزش
	0.0438	0.0444

آزمایش چهار نرخ یادگیری:

این شبکه با نرخ یادگیری ۰.۱ و بهینه ساز adam و ۵ اپیاک به کمک کتابخانه های torchbearer آموزش داده شد و به نتایج زیر رسیدیم.



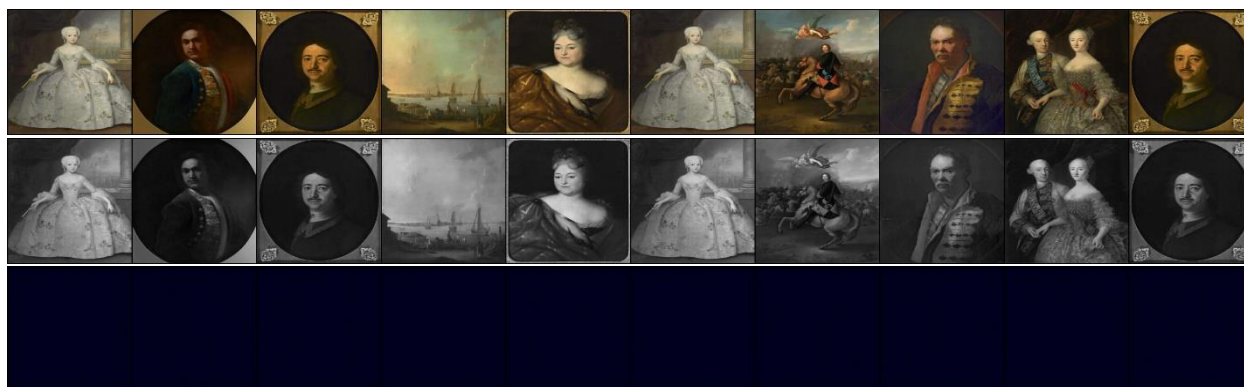
فرایند یادگیری	خطا بر روی مجموعه داده تست	خطا بر روی مجموعه داده آموزش
	0.1148	0.1225

نتیجه: با افزایش نرخ یادگیری مشاهده میکنیم در مقادیر زیاد مدل به کلی چیز خاصی یاد نمیگیرد و در لوکال اپتیمم گیر میکند و نمیتواند به درستی در فضای پارامتر حرکت کند و تصمیم میگیرد که رنگ قالب قهوه ای یا

سیاه را در تمامی پیکسل ها خروجی دهد اما در مقدار 0.001 بهترین و شارپ ترین عکس ها را خروجی گرفتیم که این یافته ها با دانسته های قبلی ما نیز همخوانی دارد و مقادیر زیاد نرخ یادگیری مشکل زا هستند و باعث میشوند که مدل در فضای پارامتر ها گم شود و نرخ های بسیار کوچک سرعت یادگیری را کاش می دهند و برای آموزش نیاز به ایپاک های بیشتری دارند اگر در لوکال اپتیمم خاصی گیر نکرده باشند.

آزمایش یک الگوریتم بهینه ساز:

این شبکه با نرخ یادگیری 0.001 و بهینه ساز SGD و 5 ایپاک به کمک کتابخانه های torchbearer آموزش داده شد و به نتایج زیر رسیدیم.

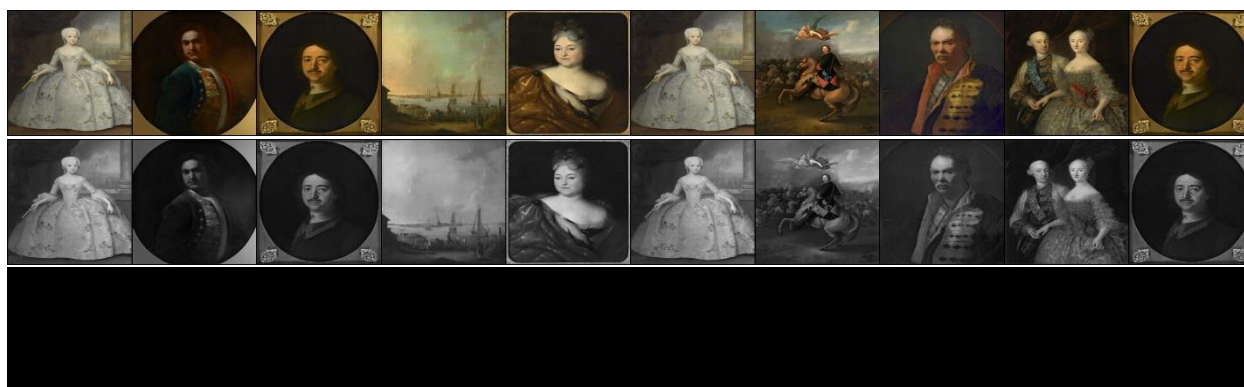


فرایند یادگیری	خطا بر روی مجموعه داده تست	خطا بر روی مجموعه داده آموزش
	0.104	0.1126

آزمایش دو الگوریتم بهینه ساز:

این شبکه با نرخ یادگیری 0.001 و بهینه ساز Adadelata و 5 ایپاک به کمک کتابخانه های torchbearer آموزش داده شد و به نتایج زیر رسیدیم.





فرایند یادگیری	خطا بر روی مجموعه داده تست	خطا بر روی مجموعه داده آموزش
	0.1137	0.1224

نتیجه: هر سه الگوریتم بررسی شده adam, sgd, adamdelta نتایج نسبتاً متفاوتی داشتند و دو الگوریتم adamdelta, sgd به کلی نتوانستند فرایند یادگیری را تکمیل کنند و عملاً فرایند یادگیری در آن‌ها رخ نداد ولی فرایند یادگیری الگوریتم sgd نشان می‌دهند که با احتمال خوبی در ایپاک‌های بیشتر نتایج بهتری می‌گیرد اما در تعداد ایپاک محدود الگوریتم adam از باقی بهتر عمل کرده است.

### معماری عمیق:

این معماری بسیار شبیه معماری قبلی است تنها تعداد لایه‌های آن افزایش یافته است و به ۵ لایه در انکدر با کانال‌های ۳۲ و ۳۲ و ۳۲ و ۳۲ و ۶۴ می‌باشد و در قسمت دیکدر نیز ۵ لایه کانولوژنی با کانال‌های ۶۴ و ۳۲ و ۳۲ و ۳۲ و ۳۲ لایه کانولوژنی وارون می‌باشد و عکس با ۱ کانال دریافت می‌کند و عکس با سه کانال خروجی می‌دهد.

این شبکه با نرخ یادگیری ۰.۰۰۱ و بهینه‌ساز adam و ۱۰ ایپاک به کمک کتابخانه‌های torchbearer آموزش داده شد و به نتایج زیر رسیدیم.



فرایند یادگیری	خطا بر روی مجموعه داده تست	خطا بر روی مجموعه داده آموزش
	0.0057	0.0062

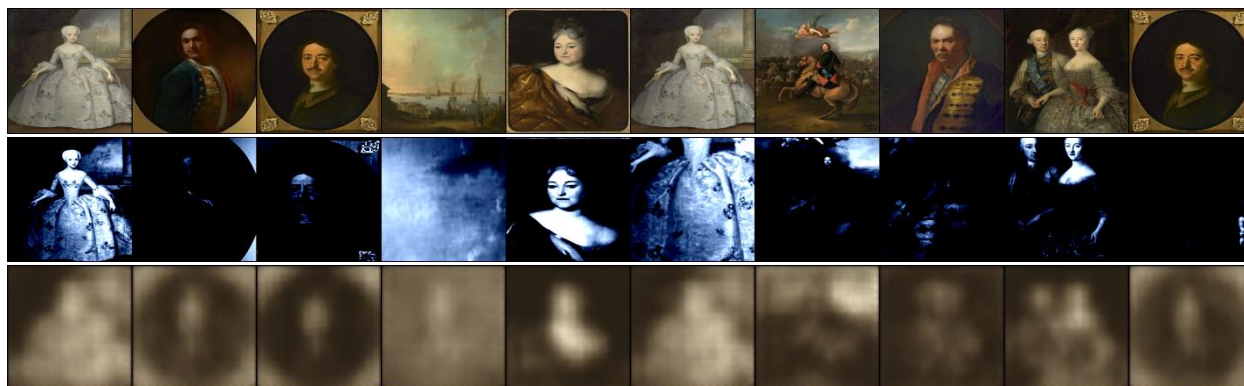
نتیاج مقداری تار تر از حد انتظار شده و این بدین دلیل است که لایه LATENT بسیار کوچک شده و مقدار زیادی از اطلاعات برای باز گردانی تصویر از دست رفته و شبکه وظیفه سخت تری را دنبال میکند، در این قسمت به ترید افی بین عمیق شدن و اندازه فضای LATENT میرسیم.

### انکدر از پیش آموزش دیده inception\_v3:

در این بخش شبکه ای طراحی کردیم که به عنوان انکدر از شبکه از پیش آموزش یافته inception\_v3 استفاده می کند و در بخش دیکدر از فضای latent لایه آخر شبکه انکدر مشابه قسمت دیکدر شبکه عمیقی که طراحی کردیم اقدام به باز سازی تصویر میکنیم. لایه های دیکدر شامل ۶ لایه کانولوزن و ارون با کانال های ۶۳ و ۳۲ و ۳۲ و ۳۲ و ۳۲ و ۳۲ است. برای اینکه انکدر ما به درستی کار کند مجبور بودیم عکس های ورودی را به ابعاد ۲۹۹ در ۲۹۹ تبدیل کنیم و با مقادیر میانگین و std برابر با (0.485, 0.456, mean=[0.406], std=[0.229, 0.224, 0.225]) کنیم و در نهایت وزن های شبکه inception\_v3 را فیریز کردیم که در فرایند یادگیری تغییر نکنند و گرادیان بر روی آن ها حرکت نکنند.



این شبکه با نرخ یادگیری ۰.۰۰۱ و بهینه ساز adam و ۱۰ اپاک به کمک کتابخانه های torchbearer آموزش داده شد و به نتایج زیر رسیدیم.



فرایند یادگیری	خطا بر روی مجموعه داده تست	خطا بر روی مجموعه داده آموزش
	0.0202	0.0251

نتایج خیلی نتایج خوبی نیستند، به دلیل اینکه کراپ های زده شده به صورت رندوم هستند شبکه اطلاعات مناسبی از ورودی ندارد که بخواهد آن را آموزش بدهد، و در نقاشی ها معمولا نقاشی های قدیمی تر و تصاویر افراد هستند و ابجکت خاصی در آن ها نیست که بتواند کمکی به آموزش ما بکند و شاید بهتر بود که وزن هارا فیریز نمیکردیم اما توان محاسباتی این امکان را به ما نمیداد و از طرف دیگر فضای latent این شبکه هم بسیار از باقی شبکه ها کوچک تر بود و باز هم با وجود این چالش های بزرگ شبکه به خوبی موفق به انجام وظیفه خود شده است و نتایج آنقدر هم بد نیست.

### معماری بر پایه U-net:

در این شبکه اسکپ کانکشن هایی از بخش انکدر به دیکدر وجود دارد که مستقیما در هنگام باز تولید تصویر به فیچر مپ ها اضافه میشود، ایده پشت این کار این است که گرادیان در شبکه های عمیق این سبکی از بین نرود و بتواند به لایه های ابتدایی از طریق این کانکشن ها برسد، و از طرفی میدانیم در لایه های کانولوشن واریان

بسیاری از اطلاعات از دست رفته در مراحل اندینگ دیگر امکان باز نشانی کامل آن ها را نداریم و در بعضی موارد باعث میشود تصاور بدست آمده شطرنجی باشند و یا شفاف نباشد این کانکشن ها این کم را به ما میکنند که از این دو مورد جلو گیری کنیم. معماری شبکه ای که ساختیم مشابه معماری کوچک است

```
x_e1 = self.encoder_l1(x)
x_e2 = self.encoder_l2(x_e1)
x_e3 = self.encoder_l3(x_e2)
x_d1 = self.decoder_l1(x_e3)
x_d1 = x_d1 + self.skip(x_e2)
x_d2 = self.decoder_l2(x_d1)
x_d2 = x_d2 + self.skip(x_e1)
x_d3 = self.decoder_l3(x_d2)
```

از لایه دوم انکدر به لایه اول دیکدر و از لایه اول انکدر به لایه دوم دیکدر اسکیپ کانکشن داریم.

این شبکه با نرخ یادگیری ۰.۰۰۱ و بهینه ساز adam و ۵ اپاک به کمک کتابخانه های torchbearer آموزش داده شد و به نتایج زیر رسیدیم.



فرایند یادگیری	خطا بر روی مجموعه داده تست	خطا بر روی مجموعه داده آموزش
	0.0035	0.0041

همانطور که قابل مشاهده است تصاویر تولید شده از شفافیت و وضوح خوبی برخوردارند.

### سوال پنجم: طراحی شبکه ای برای حذف نویز فلفل و نمک از عکس ها:

در این بخش ابتدا به صورت رندوم و با احتمالی متفاوت برای هر عکس ابتدا نویز های فلفل و نمکی (نقاط سیاه و سفید) به عکس ها وارد کردیم و در این دیتاست عکس ورودی و بردار ویژگی ورودی  $X$  عکس نویز دار و بردار خروجی و هدف  $Y$  عکس اصلی رنگی است و سپس بکمک شبکه اوتو انکدر طراحی کرده با معماری کوچک در بخش قبلی اقدام به آموزش شبکه ای با توانایی حذف نویز های فلفل نمکی از عکس ها کردیم.

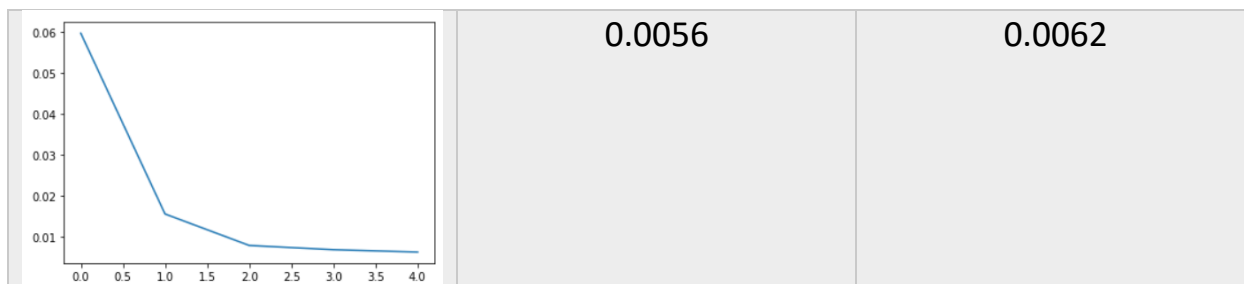


شبکه با معماری کوچک را با تغییر چنل های ورودی های آن ها از یک به سه باهمان تابع خطا آموزش دادیم.

این شبکه با نرخ یادگیری ۰.۰۰۱ و بهینه ساز adam و ۱۰ اپیاک به کمک کتابخانه های torchbearer آموزش داده شد و به نتایج زیر رسیدیم.



خطا بر روی مجموعه داده	خطا بر روی مجموعه داده	فرایند یادگیری
آموزش	تست	



نتایج نسبتاً خوب هستند ولی از شفافیت خوبی برخوردار نیستند و با توجه به نویز بسیار زیادتر از واقعیتی که به تصاویر وارد کردیم قابل درک است.

## تمرین ۲:

### مسئله پیاده سازی VAE بر روی دیتاست میوه ها

در این مسئله ۹۰۴۸۳ تصویر از ۱۳۱ میوه مختلف در ابعاد ۱۰۰ در ۱۰۰ در اختیار ما قرار گرفته به دلیل مشکلات موجود در لود کامل این مجموعه داده در کرنل های کگل تنها ۳۰ میوه از آن ها را انتخاب کردیمو یک بتا VAE را بر روی این مجموعه داده ها به کمک کتابخانه torchbearer و توانایی ایجاد کال بک های آن تابع لاس KL divergence را به تابع خطای اصلی اضافه میکنیم، شبکه ای که طراحی کردیم در قسمت انکدر ۳ لایه کانولوشنی با کانال های ۶۴ و ۳۲ و ۳۲ است و فضای latent = 5 برای مقادیر mu و logvar استفاده میکنیم و یک مرحله آپ سمپلینگ به کمک یک لایه خطی از نورون ها صورت میگیرد که به ابعاد مناسب شبکه کانولوشنی دیکدر هماهنگ شود و این بخش از شبکه شامل سه لایه کانولوشنی وارون با کانال های ۶۴ و ۳۲ و ۳۲ است.

```
class VAE_latent(nn.Module):
    def __init__(self, latent_size):
        super(VAE_latent, self).__init__()
        self.latent_size = latent_size

        self.encoder = nn.Sequential(
            nn.Conv2d(3, 32, 4, 1, 2),
            nn.ReLU(True),
            nn.Conv2d(32, 32, 4, 2, 1),
            nn.ReLU(True),
            nn.Conv2d(32, 64, 4, 2, 1)
        )

        self.mu = nn.Linear(64 * 25 * 25, latent_size)
        self.logvar = nn.Linear(64 * 25 * 25, latent_size)

        self.upsample = nn.Linear(latent_size, 64 * 25 * 25)
        self.decoder = nn.Sequential(
            nn.ConvTranspose2d(64, 32, 4, 2, 1),
            nn.ReLU(True),
            nn.ConvTranspose2d(32, 32, 4, 2, 1, 1),
            nn.ReLU(True),
            nn.ConvTranspose2d(32, 3, 4, 1, 2)
        )

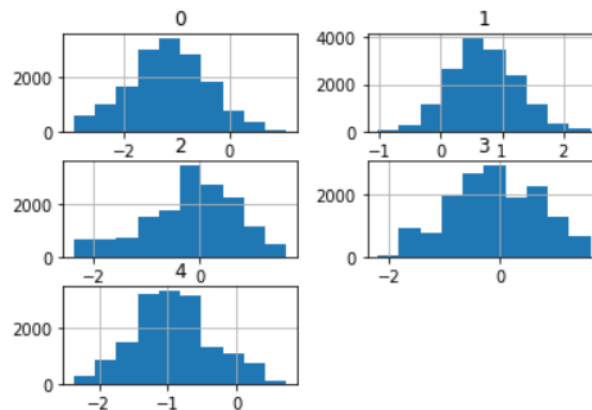
    def reparameterize(self, mu, logvar):
        if self.training:
            std = torch.exp(0.5*logvar)
            eps = torch.randn_like(std)
            return eps.mul(std).add_(mu)
        else:
            return mu
```

```
def forward(self, x):
    image = x
    x = self.encoder(x).relu().view(x.size(0), -1)
    mu = self.mu(x)
    logvar = self.logvar(x)
    return mu, logvar
```

این مدل را به کمک الگوریتم بهینه ساز adam و نرخ یادگیری ۰.۰۰۰۵ و به کمک کتابخانه torchbearer آموزش دادیم و به نتایج زیر رسیدیم.

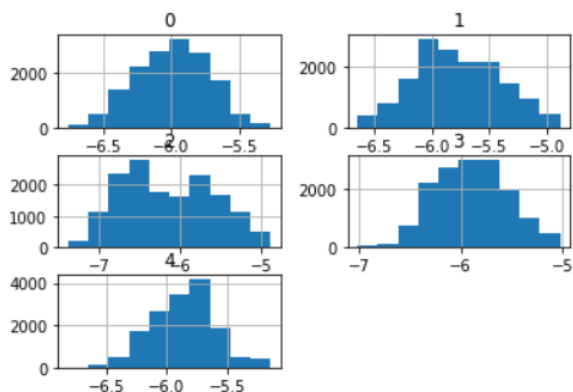
خطا بر روی مجموعه داده تست	خطا بر روی مجموعه داده آموزش
19005.3105	17458.7891

توزیع های لایه latent را برای پارامتر mu بررسی میکنیم:



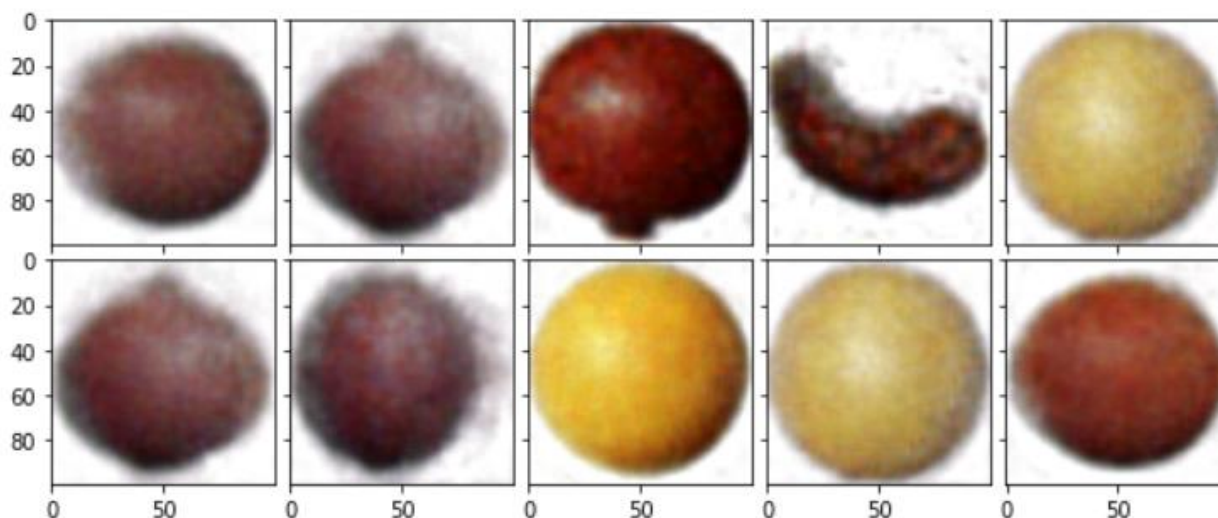


توزیع های لایه latent را برای پارامتر logvar بررسی میکنیم:



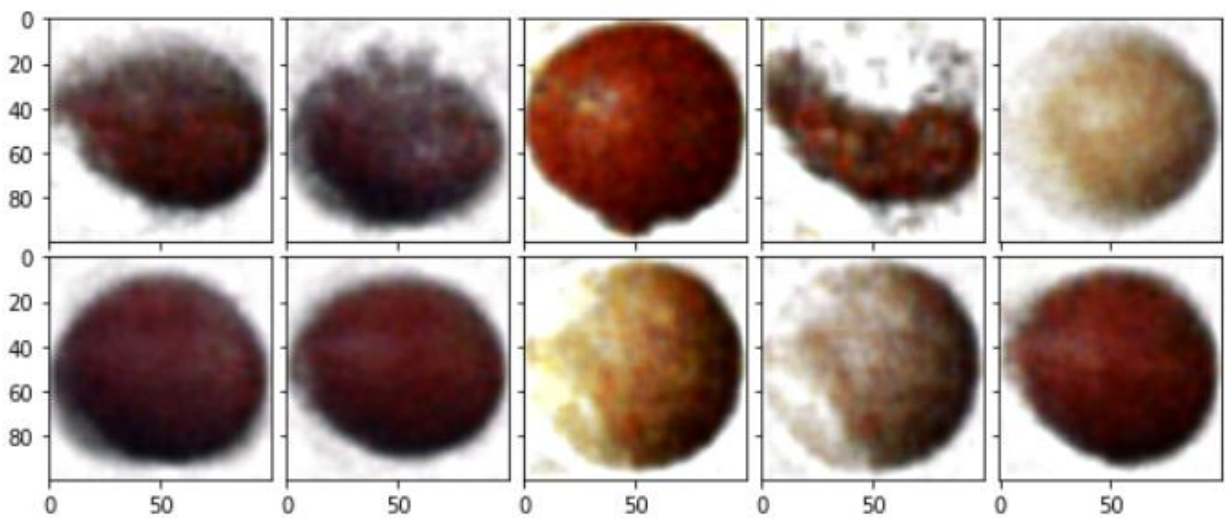
حال اگر از این توزیع ها نمونه گیری کنیم و به شبکه دیکدر بدهیم به احتمال زیاد میوه ای تازه تولید می شود.

برای بررسی این موضوع ده نمونه از میوه های تولید شده را توسط دیکدر پلات می کنیم:



حال مقداری نویز به این مقادیر اضافه میکنیم و به اصطلاح از او توزیع ها نمونه گیری میکنیم و به دیکدر پاس

میدهم تا عکس ها را برای ما تولید کند و به نتایج زیر میرسیم:



نتایج نسبتاً خوب اکثر میوه های دیتا ست گرد هستند و تا حد قابل قبولی تصاویر بازیابی شده اند.