۱-توضیح موضوع:

هدف این مقاله تولید تصاویر آسفالت ترک خورده است تا با افزایش حجم دیتاست بتوانیم مدلهای بهتری از Object Detector ها را آموزش دهیم زیرا افزایش حجم دیتا به بهبود عملکرد مدل می‌تواند کمک کند.

۲-شرح دیتاست:

تصاویر مورد استفاده به دو روش مختلف گردآوری شده است.

۲-۱-روش اول:

در این روش ابتدا دوربینی با قابلیت ثبت ویدئو با نرخ فریم ۴۰ فریم در ثانیه روی یک اتومبیل نصب می‌شود، سپس راننده شروع به حرکت می‌کند و دوربین نیز به ثبت ویدئو از سطح خیابان یا بزرگراه می‌پردازد. بعد از اتمام فرآیند فیلم‌برداری، افرادی مامور استخراج فریم‌هایی از این فیلمها می‌شوند که حاوی آسفالت ترک خورده یا آسیب‌دیده است.

۲-۲-روش دوم:

در این روش با کمک گوشی‌های موبایل اندروید و آیفون، تصاویری از سطح آسفالت‌های آسیب‌دیده ثبت می‌شود. فاصله‌ی دوربین تا سطح آسفالت بین ۱.۴ تا ۱.۶ متر است. ضمنا از سه مدل مختلف گوشی اندروید و سه مدل مختلف آیفون برای گردآوری تصاویر استفاده شده است.

تصویری از مقاله:

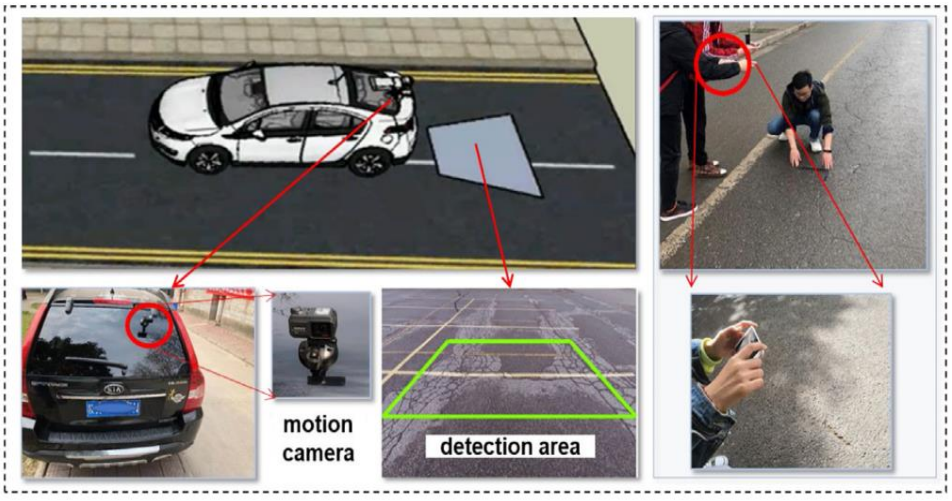


Fig. 3. Automatic videography using a vehicle-mounted motion camera.

جدول توزیع تصاویر که در مجموع ۳۰۰۰ تصویر جمع‌آوری شده است ولی حدودا ۱۴۰۰ تصویر واقعی را منتشر کرده‌اند و ۷۰۰ تصویر تولیدی خودشان را هم قرار داده‌اند.

Method	Linear Cracks	Sealed & Alligator
Motion Camera	700	505
Smartphones	995	800
Sub Total	1695	1305

در گام بعدی، این تصاویر باید پیش‌پردازش شوند تا اثرات نویز و تارشدگی و ... رفع شده و ضمنا کنتراست و روشنایی تصاویر نیز بهبود یابد:

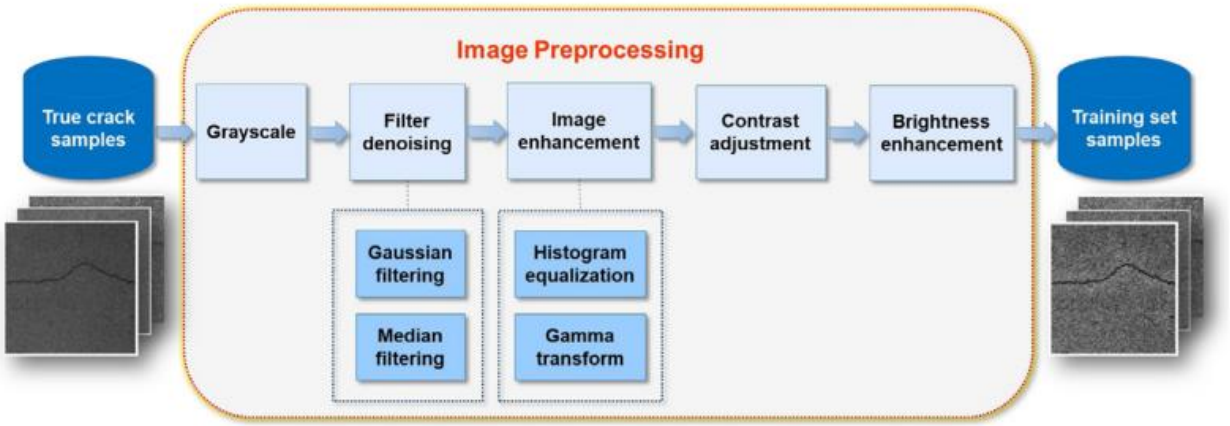


Fig. 4. Flowchart of crack image preprocessing.

درباره‌ی این مراحل در ارائه‌ی کلاسی مفصل توضیح دادیم؛ اما به طور مختصر ابتدا تصاویر سیاه سفید می‌شوند تا دامنه‌ی اطلاعات قابل یادگیری برای مدل محدود شده و سریعتر بتوانیم مدل خود را آموزش دهیم. در مرحله‌ی بعدی از فیلترهای پایین‌گذر گاوسی و میانه استفاده شده است. نویز گاوسی می‌تواند به علت خصوصیات دوربین ایجاد شود. در گام بعدی برابرسازی هیستوگرام

تصویر انجام شده است که چون تصاویر زیر نور طبیعی گرفته شده‌اند می‌تواند باعث بهبود کنتراست تصاویر شود. در مرحله‌ی بعدی نیز روشنایی تصاویر بهبود یافته است اما جزییات بیشتری از نحوه‌ی انجام این پیش‌پردازشها داده نشده است و چون تصاویر اولیه نیز منتشر نشده‌اند، نمی‌توان تخمین دقیقی از شرایط تصاویر اولیه داشت.

۳-شرح مختصر انواع خرابی آسفالت:

ترک طولی

ترک تمساحی

ترک پرشده

ترک عرضی

۴-شرح الگوریتم مورد استفاده:

در این مقاله، دو شبکه‌ی Variational Autoencoder و GAN ترکیب می‌شوند. می‌دانیم که هر دو از مدلهای زایا یا مولد هستند. ابتدا خودرمزگذار آموزش داده می‌شود و سپس با دریافت تصاویر ورودی که بین ۰ تا ۱ نرمال شده‌اند، خروجی ۱۲۸ بعدی برای ماژول مولد شبکه‌ی GAN تولید می‌کنند.

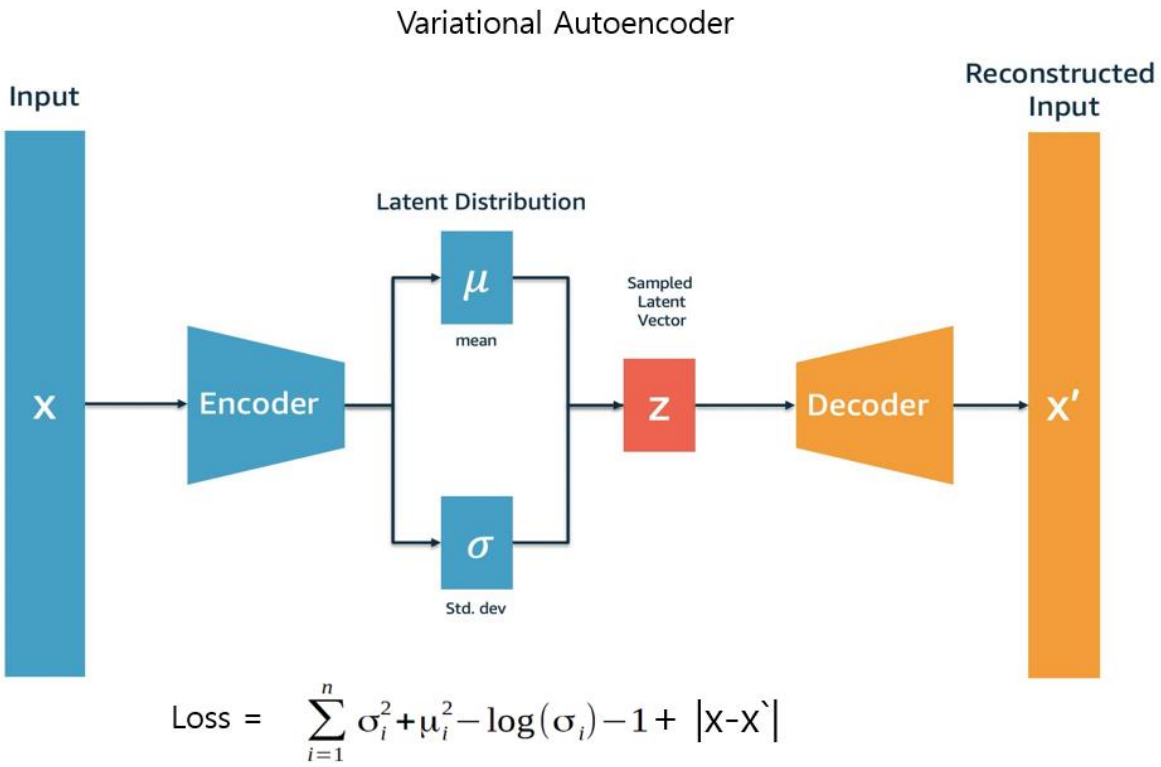
۴-۱-شرح Variational Autoencoder:

این شبکه در اصل ماهیتش تفاوتی با خودرمزگذار معمولی ندارد و یک قسمت از تابع خطایی که بر اساس آن آموزش داده می‌شود نیز مربوط به خطای بازسازی ورودی است. تفاوتش با خودرمزگذار معمولی در دو چیز است:

۱- به جای تولید یک بردار که حاوی ویژگی‌های نهان تصاویر ورودی است، دو بردار نهان تولید می‌کند که یکی نشان از میانگین به ازای هر کدام از ابعاد ویژگی است و دیگری لگاریتم واریانس. در واقع فرض ما در فرآیند آموزش این شبکه این است که تصاویر ورودی از توزیع نرمال تبعیت می‌کنند و حالا از خودرمزگذار انتظار داریم این توزیع را برای ما استخراج نماید. بعد از تولید این دو بردار، یک نمونه‌برداری به ازای هر کدام از ابعاد انجام می‌دهیم چون همیشه مدلهای زایا برای تولید دادگان جدید نیاز به Randomness دارند. در نتیجه با کمک دو بردار تولید شده توسط خودرمزگذار، یک بردار تصادفی تولید می‌شود که اختلاف این بردار با توزیع نرمال(میانگین برابر صفر و واریانس برابر ۱) قسمت دوم تابع خطای آموزش شبکه‌ی خودرمزگذار است. در واقع خودرمزگذار را مجبور می‌کنیم که از داخل دادگان ورودی خود، توزیع نرمال استخراج کند.

۲- تابع خطایش علاوه بر ترم بازسازی دادگان ورودی، ترم دیگری دارد که ذکر شد.

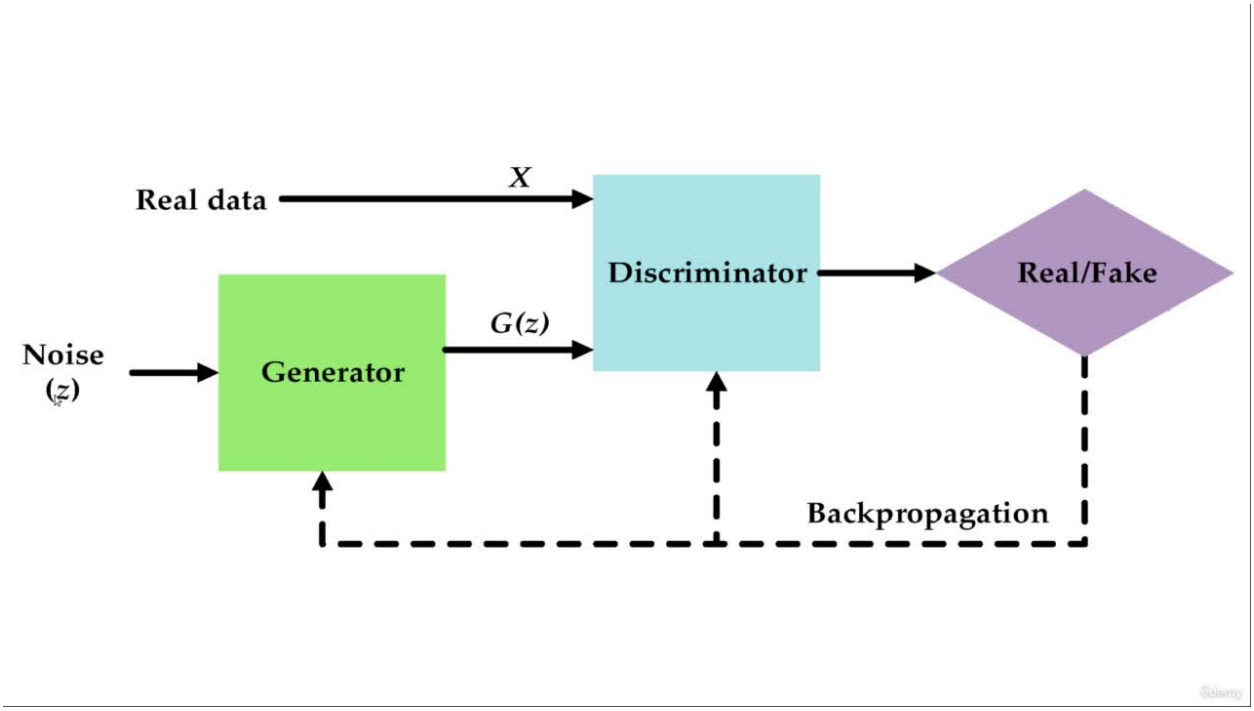
شماتیک شبکه‌ی Variational Autoencoder:



۴-۲-شرح شبکه‌ی GAN:

این شبکه از دو قسمت تولیدکننده و تفکیک‌دهنده تشکیل می‌شود. قسمت تولیدکننده با دریافت یک بردار نویز، تصویری تولید می‌کند؛ سپس این تصویر و تصویری واقعی به شبکه‌ی تفکیک‌دهنده داده می‌شود و این شبکه باید دسته‌بندی تصاویر واقعی و تصاویر جعلی تولید شده توسط تولیدکننده را انجام دهد. اگر تفکیک‌دهنده توانست درست دسته‌بندی کند، خطایی متوجه وزنهایش نمی‌شود و در غیر این صورت، با عملیات Back propagation وزنهایش تغییر می‌کنند. وزنهای تولیدکننده هم فقط در صورتی تغییر می‌کند که نتواند تفکیک‌دهنده را فریب دهد. در واقع شبکه GAN بر اساس رقابت دو شبکه آموزش داده می‌شود.

شماتیک شبکه GAN را در صفحه‌ی بعد خواهیم دید.



نوآوری مقاله در ترکیب خودرمزگذار و GAN است به این صورت که ورودی تولیدکننده‌ی GAN به جای آن که توسط ما تولید شود، توسط خودرمزگذار تولید خواهد شد.

۵-۱- معماری پیشنهادی مقاله و بهینه‌سازی‌های انجام شده توسط من:

Table 1
Generator network parameter settings.

Layer number	Type	Filter size	Stride	Output channels	Activation function
1	FC	/	/	1024	ReLU
2	Deconv	5 × 5	2	512	ReLU
3	Deconv	5 × 5	2	256	ReLU
4	Deconv	5 × 5	2	128	ReLU
5	Deconv	5 × 5	2	3	Tanh

Table 2
Discriminator network parameters.

Layer number	Type	Filter size	Stride	Output channels	Activation function
1	conv1	5 × 5	2	128	LeakyReLU
2	conv2	5 × 5	2	256	LeakyReLU
3	conv3	5 × 5	2	512	LeakyReLU
4	conv4	5 × 5	2	1024	LeakyReLU
5	FC	/	/	1	Sigmoid

در حالات مختلفی، معماری پیشنهادی مقاله را بهینه کرده‌ام. از افزودن Dropout و Batch Normalization گرفته تا تغییر Batch Size و افزودن Label Smoothing و تغییر لایه‌ها و تعداد فیلترها که در ادامه خواهیم دید.

ابتدا شبکه‌ی Variational Encoder را در سه حالت اجرا می‌کنیم. این شبکه قرار است ابتدا پیش‌آموزش داده شود و سپس با و بدون Fine tune شدن، به کار گرفته خواهد شد.

روی پیش‌آموزش شبکه‌ی خودرمزگذار چندان مانور نمی‌دهیم چون بعدا قرار است fine tune شود.

معماری خودرمزگذار حالت اول:

```
latent_dim = 128

encoder_inputs = keras.Input(shape=(224, 224, 3))
x = layers.Conv2D(32, 3, activation="relu", strides=2, padding="same")(encoder_inputs)
x = layers.Conv2D(64, 3, activation="relu", strides=2, padding="same")(x)
x = layers.Conv2D(128, 3, activation="relu", strides=2, padding="same")(x)
x = layers.Conv2D(256, 3, activation="relu", strides=2, padding="same")(x)
x = layers.Flatten()(x)
x = layers.Dense(256, activation="relu")(x)
z_mean = layers.Dense(latent_dim, name="z_mean")(x)
z_log_var = layers.Dense(latent_dim, name="z_log_var")(x)
z = Sampling()([z_mean, z_log_var])
encoder = keras.Model(encoder_inputs, [z_mean, z_log_var, z], name="encoder")
encoder.summary()
```

مقدار خطای نهایی : ۵۷۳۰

معماری خودرمزگذار در حالت دوم:

```
latent_dim = 128

encoder_inputs = keras.Input(shape=(224, 224, 3))
x = layers.Conv2D(32, 3, activation="relu", strides=2, padding="same")(encoder_inputs)
x = layers.Conv2D(64, 3, activation="relu", strides=2, padding="same")(x)
x = layers.Conv2D(128, 3, activation="relu", strides=2, padding="same")(x)
x = layers.Flatten()(x)
x = layers.Dense(256, activation="relu")(x)
z_mean = layers.Dense(latent_dim, name="z_mean")(x)
z_log_var = layers.Dense(latent_dim, name="z_log_var")(x)
z = Sampling()([z_mean, z_log_var])
encoder = keras.Model(encoder_inputs, [z_mean, z_log_var, z], name="encoder")
encoder.summary()
```

خطای نهایی: ۶۲۵۷

معماری خودرمزگذار در حالت سوم:

```
latent_dim = 128

encoder_inputs = keras.Input(shape=(224, 224, 3))
x = layers.Conv2D(64, 3, activation="relu", strides=2, padding="same")(encoder_inputs)
x = layers.Conv2D(128, 3, activation="relu", strides=2, padding="same")(x)
x = layers.Conv2D(256, 3, activation="relu", strides=2, padding="same")(x)
x = layers.Flatten()(x)
x = layers.Dense(256, activation="relu")(x)
z_mean = layers.Dense(latent_dim, name="z_mean")(x)
z_log_var = layers.Dense(latent_dim, name="z_log_var")(x)
z = Sampling()([z_mean, z_log_var])
encoder = keras.Model(encoder_inputs, [z_mean, z_log_var, z], name="encoder")
encoder.summary()
```

خطای نهایی: ۵۶۴۵

معماری سوم را برای ادامه‌ی کار انتخاب می‌کنیم.

در گام بعدی ابتدا دو حالت مختلف GAN معمولی را اجرا می‌کنیم. تفاوت دو حالت در وجود یا عدم وجود هموارسازی برچسبها است. معماری پیشنهادی مقاله را اجرا می‌کنیم:

معماری شبکه‌ی مولد

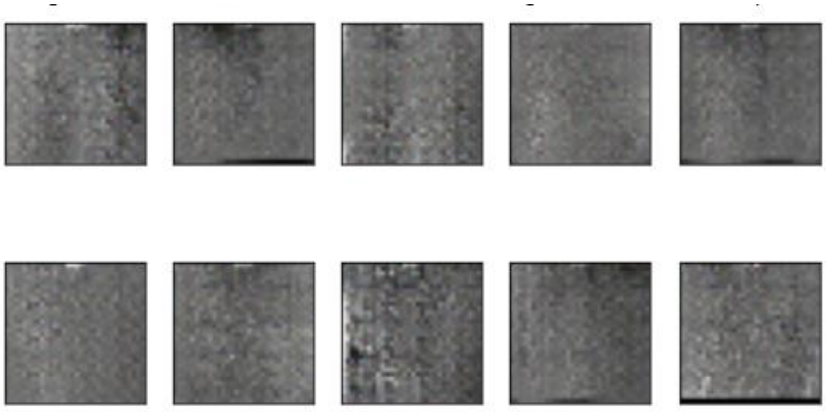
```
# latent space dimension
latent_dim = 100
X_train = discriminator_data

init = initializers.RandomNormal(stddev=0.02)

# Generator network
generator = Sequential()

generator.add(Dense(4*4*256, input_shape=(latent_dim,), kernel_initializer=init))
generator.add(Reshape((4, 4, 256)))
generator.add(LeakyReLU(0.2))

generator.add(Conv2DTranspose(128, (4, 4), strides=(2, 2), padding='same'))
generator.add(LeakyReLU(0.2))

generator.add(Conv2DTranspose(128, (4, 4), strides=(2, 2), padding='same'))
generator.add(LeakyReLU(0.2))

generator.add(Conv2DTranspose(128, (4, 4), strides=(2, 2), padding='same'))
generator.add(LeakyReLU(0.2))

generator.add(Conv2DTranspose(3, (3, 3), padding='same', activation='tanh'))
generator.summary()
```

معماری شبکه‌ی تفکیک‌دهنده

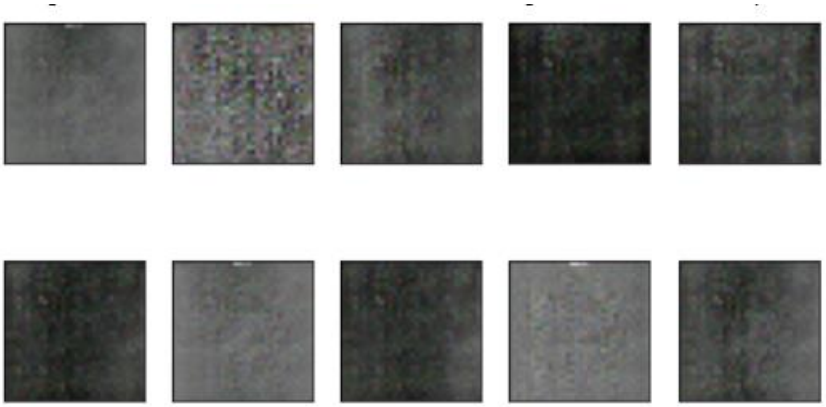
```
# Discriminator network
discriminator = Sequential()
img_shape = X_train[0].shape


discriminator.add(Conv2D(64, (3, 3), strides=(2, 2), padding='same',
                         input_shape=(img_shape), kernel_initializer=init))
discriminator.add(LeakyReLU(0.2))

discriminator.add(Conv2D(128, (3, 3), strides=(2, 2), padding='same'))
discriminator.add(LeakyReLU(0.2))

discriminator.add(Conv2D(128, (3, 3), strides=(2, 2), padding='same'))
discriminator.add(LeakyReLU(0.2))

discriminator.add(Conv2D(256, (3, 3), strides=(2, 2), padding='same'))
discriminator.add(LeakyReLU(0.2))

# FC
discriminator.add(Flatten())
discriminator.add(Dropout(0.4))
# Output
discriminator.add(Dense(1, activation='sigmoid'))
discriminator.summary()
```

در تفکیک‌دهنده بر خلاف مقاله، از Dropout استفاده کرده‌ایم چون تعداد دادگان نهایی کمتر است. نرخ smoothing برابر ۰.۱ است.

نمونه تصاویر تولید شده توسط GAN معمولی بدون Label Smooting:



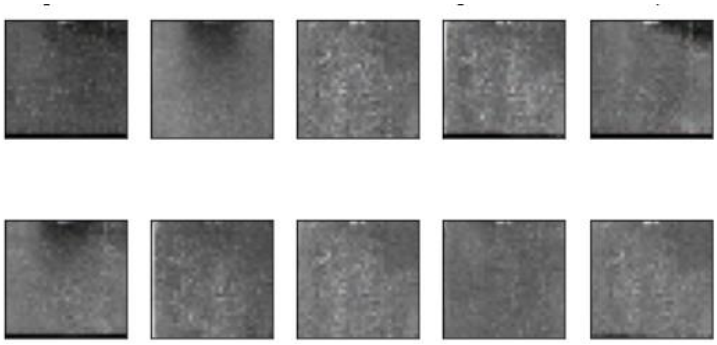
نمونه تصاویر تولید شده توسط GAN معمولی با Label Smooting:



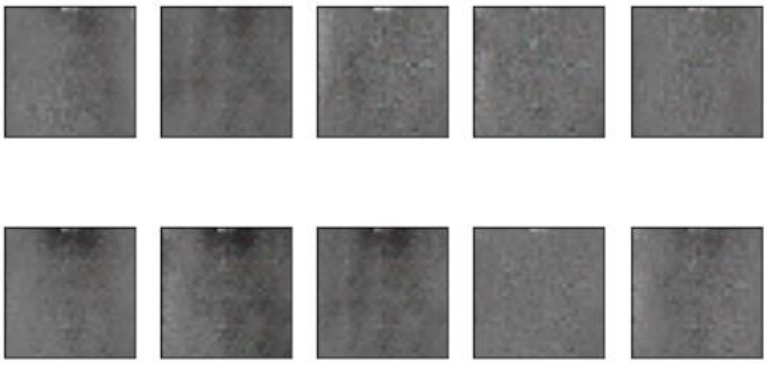
هر دو حالت ۱۰۰ گام آموزش دیده‌اند. همانطور که مشخص است میزان تارشدگی تصاویر تولیدی در حالتی که هموارسازی برچسبها را داریم کمتر است اما کنتراست تصاویر بیش از حد بالاست. این دو نمونه را صرفا به عنوان baseline در نظر می‌گیریم. با توجه به نتایج اخیر، در تمامی مدلهای آینده از هموارسازی برچسبها استفاده خواهیم کرد.

تاثیر فریز کردن یا نکردن وزنهای خودرمزگذار:

نمونه تصاویر تولید شده با معماری مقاله + Dropout + هموارسازی و فریز خودرمزگذار :



نمونه تصاویر تولید شده با معماری مقاله + Dropout + هموارسازی و عدم فریز خودرمزگذار :



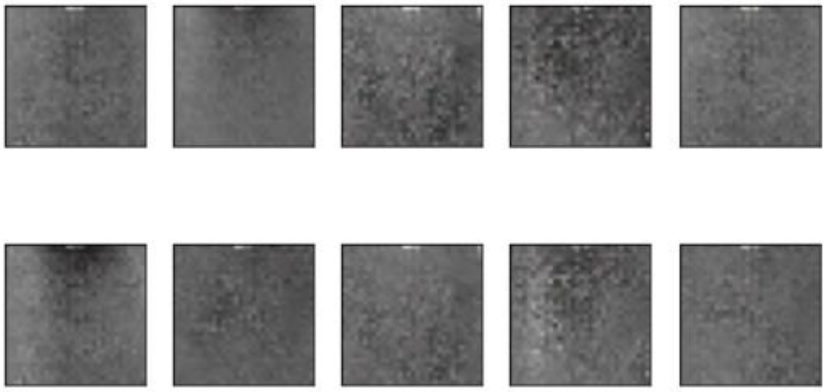
همانطور که واضح است، به صورت بصری می‌توان نتیجه گرفت که وقتی شبکه‌ی خودرمزگذار فریز نباشد می‌توان نتایج بهتری گرفت.

تاثیر افزودن Batch Normalization:

نمونه تصاویر تولید شده با Batch Normalization در کل معماری:

با مشاهده‌ی تاثیر فاجعه‌بار افزودن Batch Normalization، این ویژگی را در ادامه در نظر نخواهیم گرفت.

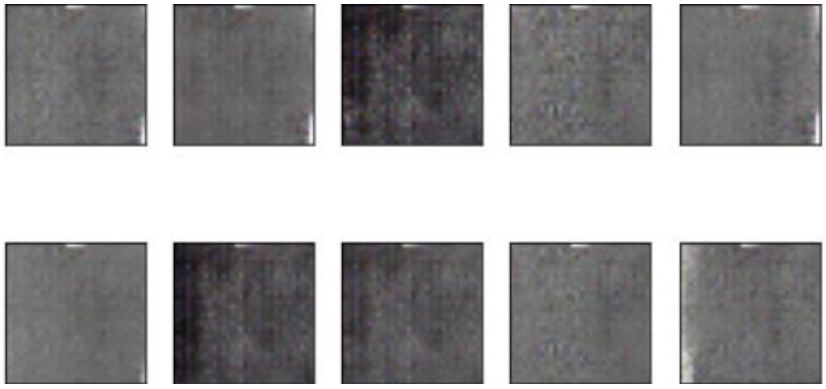
تاثیر حذف Dropout:



حذف Dropout منجر به تقویت Discriminator شده و در نتیجه generator توانایی بهبود خود را از دست می‌دهد. دقت شود که تعادل در رقابت بین این دو شبکه مهم است و با دیدن تصاویر تولید شده تصمیم به انتخاب Dropout در معماری نهایی می‌گیریم.

بررسی تاثیر نرخ Dropout:

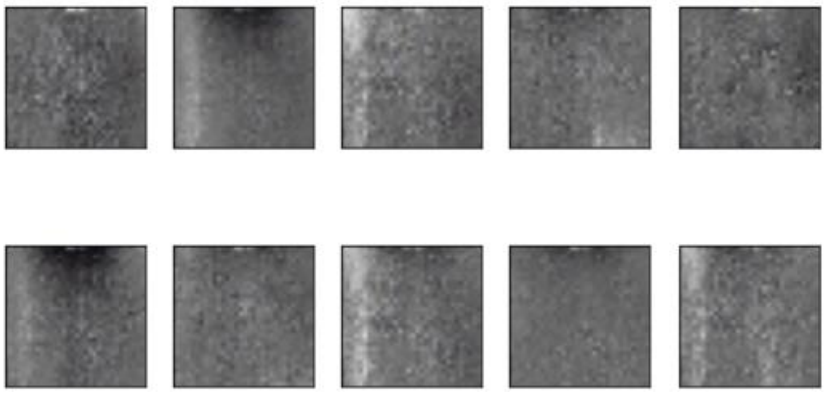
در حالات قبلی که Dropout داشتیم، نرخ آن برابر ۰.۴ بود. در این حالت، خروجی شبکه به ازای نرخ برابر ۰.۳ را بررسی خواهیم کرد:



در این حالت در گوشه‌های برخی تصاویر، نویز سفیدرنگ دیده می‌شود که مطلوب نیست و ضمنا سه تصویر بالا کنتراست بیش از حد دارند پس کاهش نرخ Dropout حداقل منجر به بهبود نتایج نمی‌شود.

در حالت بعدی نرخ Dropout را به مقدار ۰.۵ می‌رسانیم.

نمونه تصاویر تولید شده توسط شبکه:



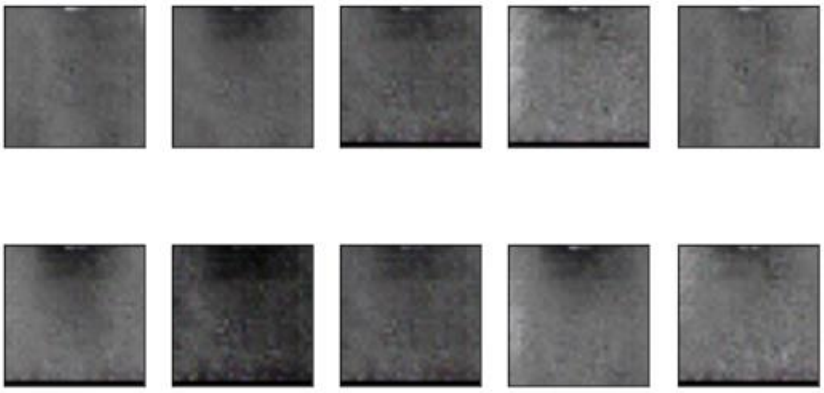
در این حالت میزان نویز در تصاویر حتی از نرخ دراپ برابر ۰.۳ نیز بیشتر است. نتیجه می‌گیریم بهتر است نه Dropout را حذف کنیم نه نرخ آن را بیشتر یا کمتر از ۰.۴ قرار دهیم.

تا اینجا متوجه شدیم بهتر است از Dropout با نرخ ۰.۴ استفاده کنیم، از نرمالسازی بچ بهره نبریم، شبکه‌ی خودرمزگذار را fine tune کنیم و از هموارسازی برچسبها هم استفاده کنیم. اینها ۴ مورد از هایپرپارامترهای شبکه هستند که تکلیفشان روشن شد.

بررسی افزایش تعداد نورونهای لایه‌ی dense شبکه‌ی generator:

در این حالت به جای ۱۰۲۴ نورون از ۲۰۴۸ نورون در لایه‌ی اول مولد استفاده می‌کنیم تا شاید تقویت مولد به تولید تصاویر بهتری منجر شود.

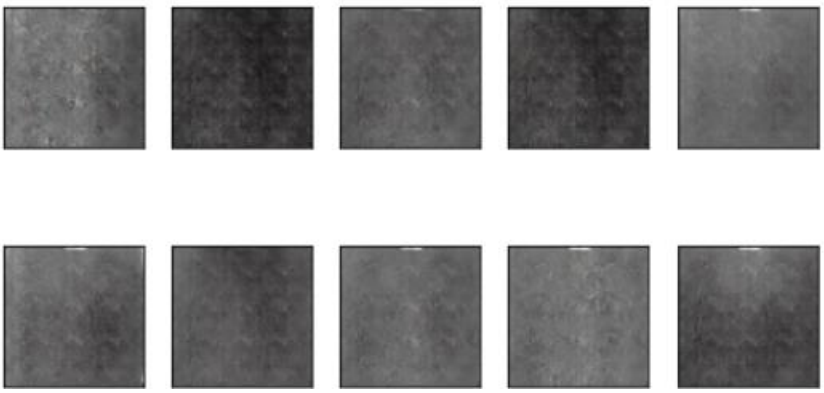
نمونه تصاویر تولید شده:



همانطور که قابل مشاهده است، افزایش نورونها منجر به تولید تصاویر با کیفیت‌تری شده است و این مدل را ذخیره می‌کنیم تا بعدا مورد ارزیابی دقیق‌تری قرار دهیم.

بررسی افزایش تعداد لایه‌های مولد:

در این حالت رزولوشن تصاویر تولیدی به جای ۳۲ در ۳۲ به ۶۴ در ۶۴ افزایش پیدا می‌کند اما فرضیه‌ای وجود دارد که اگر سایز تصاویر خیلی پایین باشد، شاید مولد نتواند اطلاعات آموخته شده‌ی خود را نمایش دهد. به همین جهت، تعداد لایه‌ها را افزایش داده و خروجی شبکه را بررسی می‌کنیم:

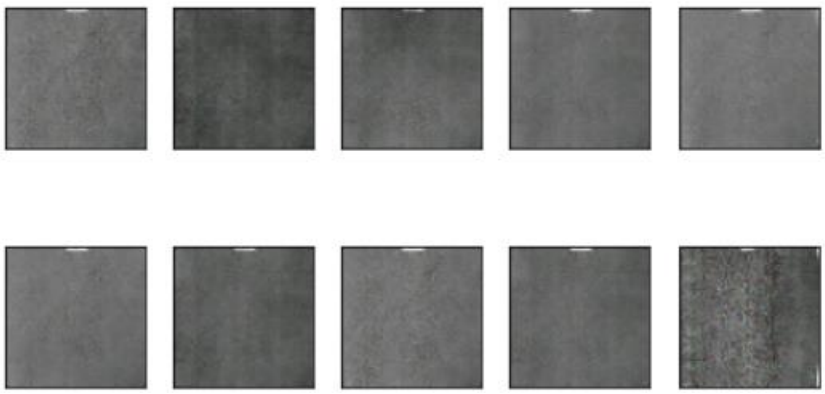


معماری مولد با لایه‌های بیشتر

```
# latent space dimension
latent_dim = 128

init = initializers.RandomNormal(stddev=0.02)

# Generator network
generator = Sequential()

generator.add(Dense(2*2*512, input_shape=(latent_dim,), kernel_initializer=init))
generator.add(Reshape((2, 2, 512)))
generator.add(ReLU())

generator.add(Conv2DTranspose(512, (5, 5), strides=2, padding='same'))
generator.add(ReLU())

generator.add(Conv2DTranspose(256, (5, 5), strides=2, padding='same'))
generator.add(ReLU())

generator.add(Conv2DTranspose(128, (5, 5), strides=2, padding='same'))
generator.add(ReLU())

generator.add(Conv2DTranspose(64, (5, 5), strides=2, padding='same'))
generator.add(ReLU())

generator.add(Conv2DTranspose(3, (5, 5), strides=2,padding='same', activation='tanh'))
generator.summary()
```

تصاویر تولید شده توسط این معماری نیز آنقدر خوب هستند تا گزینه‌ی ذخیره شدن فایل این مدل برای ارزیابی کمی و دقیق‌تر باشند.

در حالت بعدی به جای استفاده از ۶۴ فیلتر در لایه‌ی آخر deconvolution، از ۱۲۸ عدد استفاده می‌کنیم. نمونه تصاویر تولید شده:



این حالت نیز تصاویر خوبی تولید می‌کند و آن را ذخیره می‌کنیم. همانطور که قابل مشاهده است تصاویر نویز کمی دارند و در برخی، خرابی‌های نسبتا مشهودی در سطح آسفالت دیده می‌شود که برای ما مطلوب است. تمامی سه مدل اخیر که فایلشان را برای ارزیابی دقیق‌تر ذخیره کردیم از نرخ دراپ‌اوت ۰.۴ استفاده می‌کردند. دو مدل دیگر را نیز به عنوان baseline بدون دراپ و با دراپ ۰.۵ ذخیره کرده‌ایم. بنابراین ۵ مدل را در این مرحله ذخیره می‌کنیم. مجموعه‌ای شامل ۷۰۰ تصویر را نیز خود مقاله ارائه داده است که شبکه‌ی آنها تولید نموده است. بنابراین برای مقایسه‌ی کمی، ۶ حالت مختلف را خواهیم داشت.

معرفی معیارهای ارزیابی FID و IS:

معیار FID تفاوت دو توزیع احتمال حاصل از ویژگی‌های خروجی شبکه‌ی مشهور Inception را به دست می‌دهد. این دو توزیع هم حاصل از دو مجموعه تصویر هستند، یکی تصاویر واقعی و دیگری تصاویر تولید شده توسط مدل. هر مجموعه تصویر را به مدل داده و خروجی لایه‌ی ماقبل آخر که یک بردار ۲۰۴۸ بعدی است در نظر می‌گیریم. میانگین و ماتریس کوواریانسهای این بردارها را محاسبه می‌کنیم بنابراین بردار میانگین یک بردار ۲۰۴۸ بعدی و ماتریس کوواریانس هم یک ماتریس ۲۰۴۸ در ۲۰۴۸ بعدی خواهد بود سپس با کمک رابطه‌ی زیر، معیار FID را حساب می‌کنیم که اگر دو مجموعه تصویر عینا برابر باشند، مقدار آن صفر خواهد بود ولی حد بالا ندارد و هر چه کمتر باشد بهتر است.

$$FID = \|\mu_r - \mu_g\|^2 + T_r(\Sigma_r + \Sigma_g - 2(\Sigma_r\Sigma_g)^{1/2})$$

میانگین روی ابعاد برای تصاویر واقعی

میانگین روی ابعاد بر ای تصاویر تولیدی

ماتریس کوواریانس تصاویر واقعی

ماتریس کوواریانس تصاویر تولیدی

معیار Inception Score یا به اختصار IS بر خلاف FID نیاز به دو مجموعه تصویر ندارد و صرفا تنوع و کیفیت تصاویر تولیدی را بدون هیچ مرجعی می‌سنجد. در این معیار نیز از خروجی‌های مدل Inception استفاده می‌کنیم منتهی خروجی‌های لایه‌ی آخر. در این معیار P(y|x) برابر خروجی شبکه‌ی Inception به ازای یک تصویر تولید شده است که یک بردار ۱۰۰۰ بعدی شامل ۱۰۰۰ احتمال است. هر عدد نشانه‌ی احتمال تعلق تصویر ورودی به یکی از هزار کلاسی است که Inception روی آن آموزش دیده است. پس از محاسبه‌ی تمام احتمالات شرطی، میانگین آنها را به ازای تمام کلاسها محاسبه می‌کنیم و بدین ترتیب P(Y) را به دست می‌آوریم. هدف این معیار این است که KL-Divergence این دو احتمال را محاسبه کند یعنی فاصله‌ی این دو توزیع احتمال را به دست آورد. هر چه مجموعه تصاویر تولید شده بین کلاسهای مختلف بیشتر balanced باشند یعنی P(Y) به توزیع uniform نزدیکتر است و هر چه تصاویر تولید شده را بهتر بتوان به یکی از کلاسها متعلق دانست، یعنی تصاویر تولید شده به راحتی قابل تفکیک به یکی از کلاسها هستند. هر چه KL-Divergence این دو بالاتر باشد یعنی تصاویر تولید شده باکیفیت‌تر هستند پس هر چه IS بالاتر باشد بهتر است.

ضعف هر دوی این معیارها در این است که ما از خروجی‌های Inception استفاده می‌کنیم در حالی که این شبکه روی دادگان آسفالت یا آسفالت ترک‌خورده آموزش ندیده است منتهی چون معیارهای مشهوری هستند و مقاله هم از آنها استفاده کرده است، ما نیز در جدولی نتایج را می‌آوریم.

$$\mathrm{IS}(G) = \exp\left(\mathbb{E}_{\mathbf{x}\sim p_g} D_{KL}(p(y|\mathbf{x}) \parallel p(y)) \right),$$

سه مدل اول در قسمت مولد سه لایه دیکانولوشن قبل از دیکانولوشن آخر که تصویر نهایی را می‌سازد دارند و تفاوتشان در مقدار یا عدم وجود دراپ‌اوت است. دو مدل آخر(۴و۵) پیش از دیکانولوشن آخر، چهار لایه دیکانولوشن دارند. عدد اول که ۲۰۴۸ یا ۱۰۲۴ در سایر حالات است نیز تعداد نورونهای لایه دنس اول هستند. قبلا در متن گزارش با شرح جزییات کامل تمام این مدلها را معرفی نمودیم و اینجا صرفا خلاصه کرده‌ایم:

Our 1	4 layers with dropout = 0.5
Our 2	4 layers without dropout
Our 3	4 layers with dropout = 0.4
Our 4	5 layers(2048,512,256,128,64), dropout = 0.4
Our 5	5 layers(2048,512,256,128,128), dropout = 0.4

نتایج دو معیار FID و IS :

منظور از Lecture، ارزیابی تصاویر تولیدی مقاله است که نویسندگان مقاله منتشر کرده‌اند و با اعداد اعلامی آنها در متن مقاله تفاوت فاحشی دارند! شاید چون تمامی تصاویر را منتشر نکرده‌اند.

Model	**IS**	**FID**
Lecture	**2.23**	**100**
Our 1	1.47	417
Our 2	**1.53**	761
Our 3	1.50	448
Our 4	1.51	**407**
Our 5	1.43	632

با مشاهده‌ی این نتایج، از بین معماریهای ما بهترین معماری معماری چهارم است.

ارزیابی عملی:

شاید بهترین روش ارزیابی کیفیت تصاویر تولید شده بررسی تاثیر استفاده از این تصاویر در کنار تصاویر واقعی در عملکرد یک مدل Object Detection است. برای این امر، مقاله مدل Faster RCNN را انتخاب کرده است که ما نیز با پایتورچ آن را پیاده و تست کرده‌ایم. برای این کار چون امکان و زمان برچسب زدن ۲۰۰۰ تصویر را نداشتیم، صرفا در دو حالت با یا بدون استفاده از تصاویر تولیدی مقاله که برچسب هم دارند در کنار تصاویر واقعی، Faster RCNN را آموزش می‌دهیم و مانند مقاله معیار Average Precision را گزارش می‌کنیم. در هر دو حالت ۱۵ درصد دادگان را به عنوان دادگان تست و ۸۵ درصد را برای آموزش استفاده می‌کنیم.

Average Precision بدون تصاویر تولیدی: ۷۶.۵ ٪

Average Precision با تصاویر تولیدی: ۷۷.۸ ٪

همانطور که قابل مشاهده است، ۱.۳ درصد بهبود عملکرد پس از افزودن تصاویر تولیدی به تصاویر واقعی دیده می‌شود که تقریبا با اعداد اعلامی در مقاله تطابق دارد.