سوال ۱)

برای تبدیل داده های به صورت متن یا رشته به بردار از bag-of-word (BOW) استفاده می کنیم. در این روش با شمارش تعداد دفعاتی که هر کلمه ظاهر می شود، متن دلخواه را به بردارهایی با طول ثابت تبدیل می کنیم.

مثال: سه داده متنی داریم.

A: The book
B: The book on the table
C: The book on the ground
Vocab = { the , book , on , table, ground}

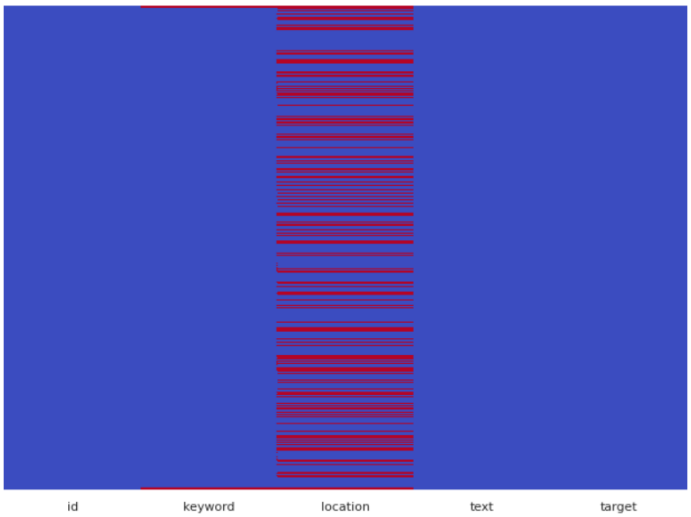
	the	*book*	*on*	*table*	*ground*
A	1	1	0	0	0
B	2	1	1	1	0
C	2	1	1	0	1

A = [1 , 1 , 0 , 0, 0]
B = [2, 1 , 1 , 1 , 0]
C = [2, 1 , 1 , 0 , 1]

سوال ۲)

```
sns.set(rc={'figure.figsize':(11,8)})
sns.heatmap(train_data.isnull(),yticklabels=False,cbar=False,cmap="coolwarm")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbbae731c10>
```

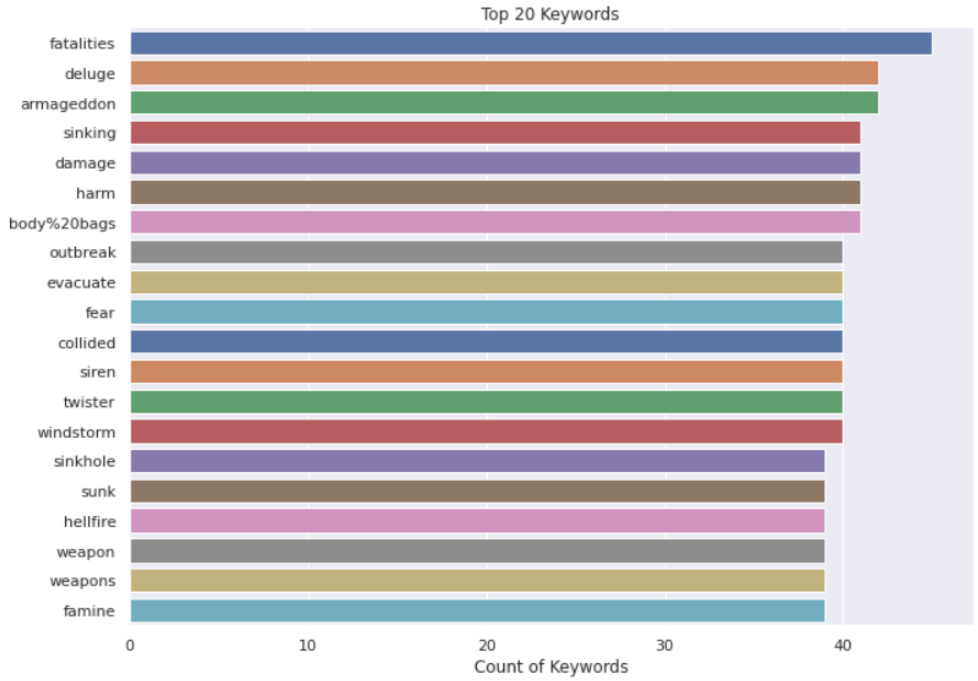


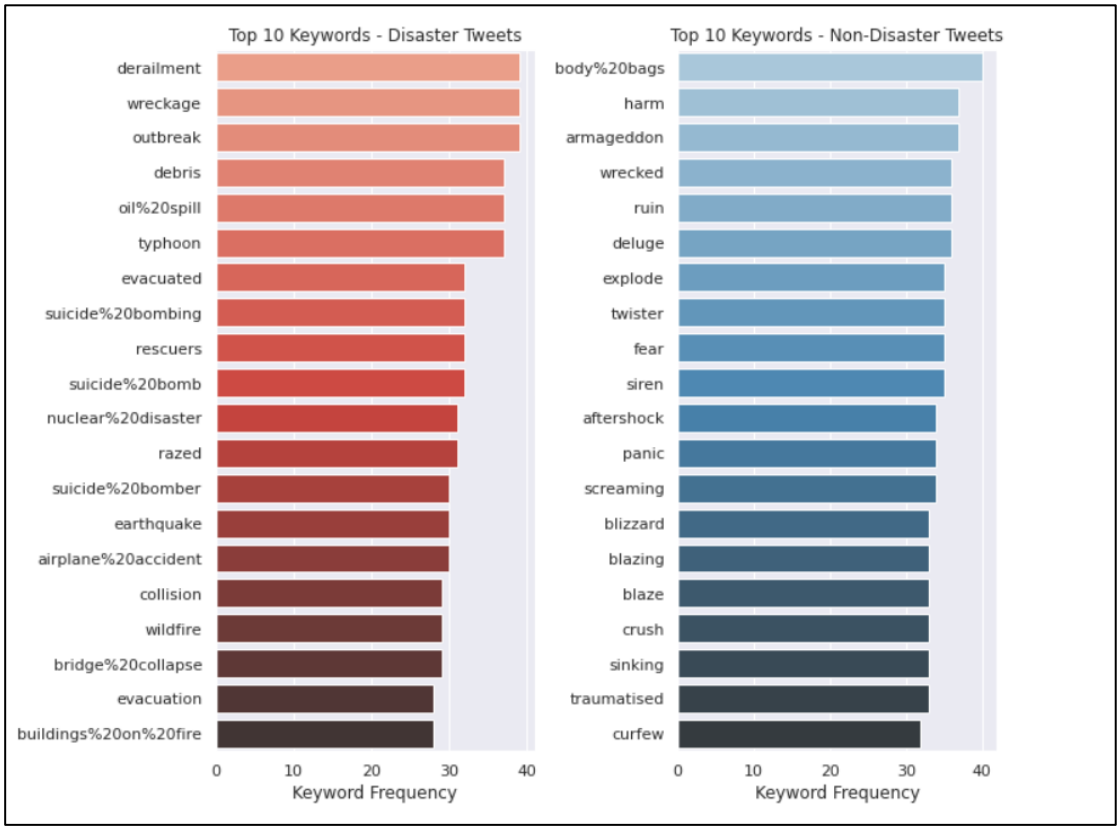
نمودار heatmap تعداد دیتاهای گمشده یا همان loss data را نشان می دهد. با دستور زیر این نمودار را میتوان رسم کرد. همانطور که میبینیم در ستون location بیشترین میزان دیتای گمشده را داریم. در این سوال ستون هایی که میخواهیم با اونا کار کنیم text و target هستند و چون در نمودار هیت مپ برای این دو ستون خط قرمزی مشاهده نمیکنیم. یعنی این دو ستون loss data ندارند و نیازی نیست پیش پردازش برای پر کردن سطرهای null انجام شود.

سوال ۳)

```
chains=train_data['keyword'].value_counts()[:20]
sns.barplot(x=chains,y=chains.index,palette='deep')
plt.title("Top 20 Keywords")
plt.xlabel("Count of Keywords")
```

```
Text(0.5, 0, 'Count of Keywords')
```





سوال ۴)

```
def toclean_text(text):

  clean_text=text.translate(str.maketrans('', '', string.punctuation))
  return clean_text

train_data['clean_text'] = train_data["text"].apply(toclean_text)
```

تابع toclean_text را برای حذف علایم نگارشی تعریف کردیم و یک ستون toclean_text که شامل ستون text بدون علایم نگارشی هست را به دیتاست اضافه کردیم.

سوال ۵)

با استفاده از re.compiler تابع حذف موارد خواسته شده ، در فایل ipynb تکمیل شد.

سوال ۶)

به کمک ابزار nltk ، و توکنایز کردن تابعی نوشتیم که stopword ها را حذف میکند. در نهایت ستون rejoined_text را به دیتاست اضافه کردیم که پیش پردازش شده داده های ورودی هست. یعنی علایم نگارشی، موارد گفته شده در سوال ۵ و stopword ها حذف شده. کد مربوط به این بخش در نوت بوک تکمیل شده.

سوال ۷)

tokenization روشی برای تفکیک یک متن خاص به تکه ها یا توکن های کوچک است.

از کلاس Tokenizer Keras برای برداری کردن یک مجموعه متن استفاده می شود .برای این کار، هر ورودی متنی به دنباله اعداد صحیح یا بردار تبدیل می شود که برای هر کلمه یک ضریب باینری (صفر یا یک) داده میشود.

اگر از text_to_sequence استفاده کنیم به دنباله ای از اعداد صحیح تبدیل می شود.
اگر از text_to_matrix استفاده کنیم به ماتریس با درایه های باینری تبدیل میشود.
دلیل استفاده: tokenization به درک متن و تفسیر معنی متن با آنالیز کردن دنباله ای از کلمات و توسعه مدل کمک میکند. مزیت توکن سازی این است که متن را به قالبی تبدیل می کند که راحت تر به اعداد خام تبدیل می شود و در واقع می توان از آن برای پردازش استفاده کرد.

```
#Define tokenizer and padding

tweet = train_data.loc[:,train_data.columns=='rejoined_text'].values.flatten()

tokenizer= Tokenizer(num_words=5000)
def tokenize(text):
    seqs = tokenizer.fit_on_texts(text)
    return seqs

tokens=tokenize(tweet)

X_train = tokenizer.texts_to_sequences(tweet)
vocab_size = len(tokenizer.word_index) + 1  # Adding 1 because of reserved 0 index
```

از keras tokenizer استفاده کردیم و تعداد توکن ها را تا ۵۰۰۰ کلمه در num_words محدود کردیم.
یعنی حداکثر ۵۰۰۰ کلمه را بر اساس فراوانی کلمات(متداول ترین کلمه ها) ، نگه دارد.
دیتا ورودی rejoined_text که برای تابع tokenize دادیم، از قبل lowercase شده بود. اگر از قبل انجام نداده بودیم میتوانستیم بعد از num_words ویژگی lower=True را هم به توکنایزر اضافه کنیم.

- فرق محدود کردن تعداد توکن و محدود نکردن آن:

برای مثال دیتای پیش پردازش در اندیس ۱ `forest fire near la ronge sask canada` است.

در صورت نداشتن محدودیت برای تعداد توکن دنباله مربوط به این دیتا به این صورت است:

```
[102, 5, 133, 548, 6445, 6446, 1215]
```

در صورت اضافه کردن num_words=5000 دنباله آن به صورت زیر خواهد بود:

```
[102, 5, 133, 548, 1215]
```

همانطور که میبینیم بعد از اضافه کردن محدودیت، طول دنباله ۲تا کم شد، آن کلماتی که جز ۵۰۰۰ کلمه متداول نبودند حذف شده اند.

- Lowercasing

بهتر است که در پیش پردازش lowercasing انجام شود، زیرا در صورت عدم انجام، دو کلمه یکی با حرف بزرگ و دیگری با حرف کوچک ،به عنوان دو کلمه متفاوت در مدل فضای برداری نمایش داده می شوند و در نتیجه باعث افزایش بعد میشود.

سوال ۸)

در مرحله قبل که text_to_sequence زدیم، هر دنباله که تولید میشود در بیشتر مواقع دارای طول کلمات متفاوتی است. برای رفع این مساله، می‌توان از pad_sequence استفاده کرد که به سادگی توالی کلمات را با صفر لایه‌گذاری می‌کند. یعنی به آخر یا اول دنباله ها صفر اضافه میکند تا همه دنباله ها سایز یکسان داشته باشند. Pad_sequence پارامتر maxlen هم دارد که برای تعیین اینکه دنباله‌ها چقدر طولانی باید باشند مورد استفاده قرار دهد.

```
maxlen = 100

X_train = np.array(X_train, dtype=object)

X_train = keras.preprocessing.sequence.pad_sequences(X_train ,padding='post', maxlen=maxlen)
```

سوال ۹) لایه embedding ما را قادر می سازد تا هر کلمه را به یک بردار طول ثابت با اندازه تعریف شده(input length) تبدیل کنیم. بردار حاصل یک بردار متراکم با مقادیر واقعی به جای ۰ و ۱ است. طول ثابت بردارهای کلمه به ما کمک می کند تا در کنار ابعاد کاهش یافته، کلمات را به شیوه ای بهتر نمایش دهیم. برای انتخاب سایز لایه جاسازی (output dim) میتوان با آزمون و خطا به بهترین نتیجه رسید، اما نکات زیر را هم میتوان در نظر داشت:

این لایه فشرده‌سازی ورودی است، وقتی لایه کوچک‌تر باشد، ورودی ها بیشتر فشرده میشوند و داده‌های بیشتری را از دست می‌دهیم. وقتی لایه بزرگ‌تر باشد ، کمتر فشرده می‌شود و ممکنه روی مجموعه داده ورودی overfit اتفاق بیفتد.

هرچه دایره لغات بزرگتری داشته باشیم و بخواهیم نمایش بهتری از آن داشته باشیم میتوان لایه را بزرگتر کرد.

اگر داده ورودی outlier زیادی دارد و بخواهیم از شر کلمات غیر ضروری خلاص شوید ، باید بیشتر فشرده کنیم یعنی سایز embedding را کوچکتر کنیم.

```
#creating model

embedding_dim = 50

model = Sequential()
model.add(Embedding(input_dim=vocab_size,
                           output_dim=embedding_dim,
                           input_length=maxlen))
model.add(layers.GlobalMaxPool1D())
model.add(layers.Dense(10, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
model.summary()
```

سوال ۱۰) ابتدا تابع plot_cm رو تعریف کردیم که confusion matrix را رسم کند.

سپس بعد از فیت کردن مدل، پیش‌بینی را روی داده های آموزش انجام دادیم و به کمک مقادیر واقعی و پیش‌بینی شده نمودار دقت و confusion ماتریس را رسم کردیم. (بررسی نتایج بدست آمده در ادامه توضیح داده میشود.)

```
# Showing Confusion Matrix
def plot_cm(y, y_pre, title, figsize=(5,5)):
    cm = confusion_matrix(y, y_pre, labels=np.unique(y))
    cm_sum = np.sum(cm, axis=1, keepdims=True)
    cm_perc = cm / cm_sum.astype(float) * 100
    annot = np.empty_like(cm).astype(str)
    nrows, ncols = cm.shape
    for i in range(nrows):
        for j in range(ncols):
            c = cm[i, j]
            p = cm_perc[i, j]
            if i == j:
                s = cm_sum[i]
                annot[i, j] = '%.1f%%\n%d/%d' % (p, c, s)
            elif c == 0:
                annot[i, j] = ''
            else:
                annot[i, j] = '%.1f%%\n%d' % (p, c)
    cm = pd.DataFrame(cm, index=np.unique(y), columns=np.unique(y))
    cm.index.name = 'Actual'
    cm.columns.name = 'Predicted'
    fig, ax = plt.subplots(figsize=figsize)
    plt.title(title)
    sns.heatmap(cm, cmap= "YlGnBu", annot=annot, fmt='', ax=ax)
```

سوال ۱۱)

اهمیت اندازه learning rate:

نرخ یادگیری زیاد به مدل اجازه می‌دهد تا سریع‌تر یاد بگیرد ولی ممکن است نقاط بهینه بدست آمده local باشد. نرخ یادگیری کمتر ممکن است به مدل اجازه می دهد نقاط بهینه را در global بدست اورد ، اما ممکن است آموزش به میزان قابل‌توجهی طول بکشد.

Activation function در لایه dense :

توابع sigmoid به طور کلی در classification بهتر عمل می کنند.

توابع sigmoid و tanh گاهی اوقات به دلیل مشکل vanishing gradient کمتر استفاده می شود.

تابع ReLU یک تابع فعال سازی عمومی است و فقط باید در لایه های مخفی استفاده شود.

اگر در شبکه‌های خود با dead neuron مواجه شدیم، تابع leaky ReLU بهترین انتخاب است.

بطور کلی، می‌توان با استفاده از تابع ReLU شروع کرد و در صورتی که ReLU نتایج بهینه را ارائه ندهد، به سراغ سایر توابع رفت.

در این تمرین ما مدل برای لایه dense اول ReLU و برای دومی sigmoid استفاده کردیم که در ادامه عملکرد انهارا بررسی میکنیم.

Optimizer:

اگر بخواهیم شبکه عصبی را در زمان کمتر و کارآمدتر آموزش دهیم، بهترین optimizer، Adam است. اگر داده های پراکنده داشتیم بهتر است از بهینه سازهای با نرخ یادگیری پویا استفاده کنیم. اگر بخواهید از GDاستفاده کنیم، mini_batch GD بهترین گزینه است.

Loss function:

Binary crossentropy متداول ترین تابع ضرر است که برای مسائل classification که دارای دو کلاس هستند استفاده می شود. در مدل ما نیز از این تابع هزینه استفاده شده.

سوال ۱۲)

برای این تمرین چندین مدل ارائه شد که نوع معماری آنها در فایل ipynb موجود است.

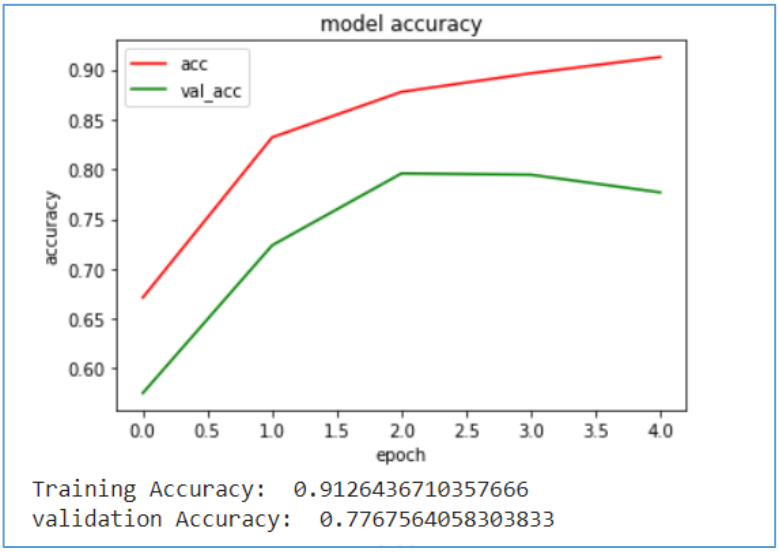
مدل اول RNN ساده بود.

مدل دوم لایه LSTM اضافه شد

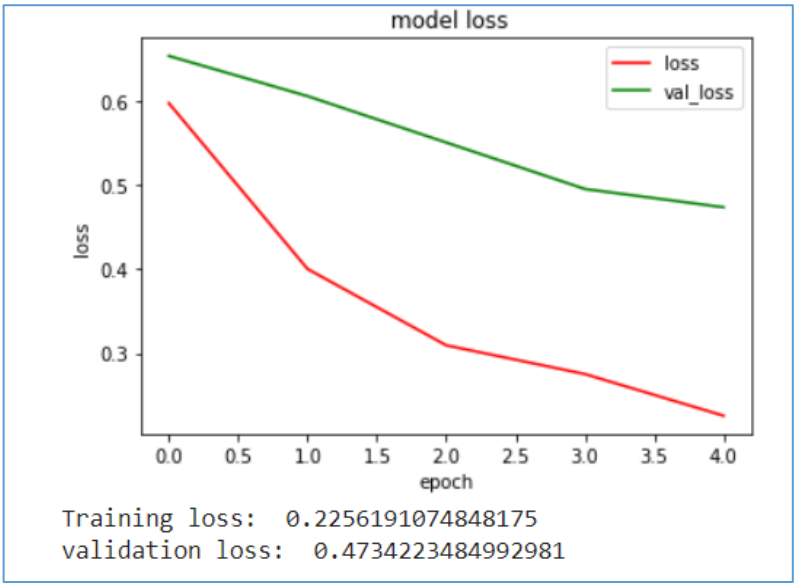
مدل سوم لایه GRU اضافه شد.

در مدل بعدی هر دو لایهLSTM و GRU اضافه شدند.

از بین این ۴تا مدلی که فقط لایه GRU اضافه شده بود، عملکرد بهتری نسبت به بقیه داشت. پس مدل بعدی را با تغییراتی برای کاهش اورفیت نوشتیم که نتایج آن را بررسی میکنیم:

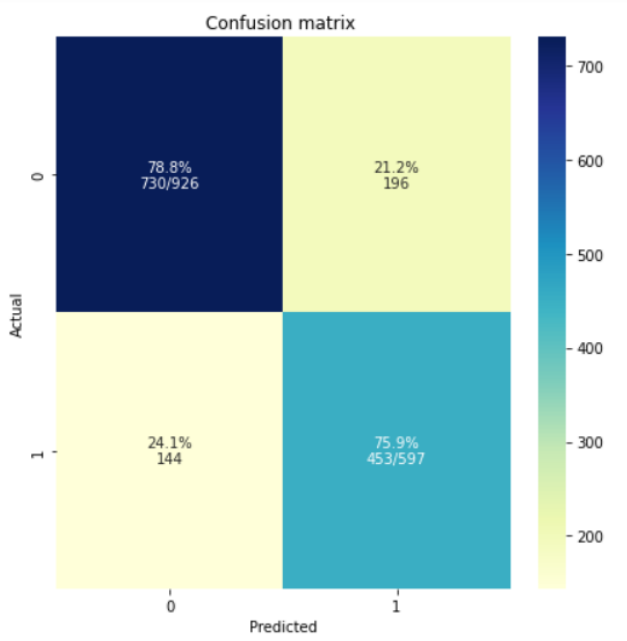


این مدل برای داده های آموزش دقت ۹۱ درصد و برای valiation دقت ۷۷درصد داشت. همانطور که در نمودار میبینیم از ایپاک تقریبا ۲ ، مدل شروع به اورفیت شدن میکند. چون علیرغم بالا رفتن دقت آموزش ، دقت داده های validation با شیب کمی رو به کاهش است.



مقدار loss در داده آموزش ۲۲ درصد و برای validation ۴۷ درصد هست.

(0=non_disaster // 1=disaster)



TP=true positive =78.8%

یعنی تعداد داده هایی که مقدار تارگت آنها ۰ بود و مدل ما نیز آنها را ۰ پیش بینی کرده.

FN=false negative =21.1%

یعنی تعداد داده هایی که مقدار تارگت آنها 0 بود و مدل ما به اشتباه آنها را 1 پیش بینی کرده.

FP= false positive =24.1%

یعنی تعداد داده هایی که مقدار تارگت آنها 1 بود و مدل ما به اشتباه آنها را 0 پیش بینی کرده.

TN= true negative =75.9%

یعنی تعداد داده هایی که مقدار تارگت آنها 1 بود و مدل ما نیز آنها را 1 پیش بینی کرده.

سوال ۱۳)

شبکه عصبی عمیق و شبکه عصبی Recurrent مشکل vanishing gradient را دارند.

در RNN این مشکل ناشی از یک سری طولانی از ضرب مقادیر کوچک(وزنها) است، که گرادیان ها را کاهش می دهد و باعث انحطاط فرآیند یادگیری می شود.

سوال ۱۴)

برای رفع overfitting:

- پیش پردازش داده های ورودی
- اضافه کردن لایه dropout . از 0.1 میتوان شروع کرد.
- اضافه کردن callback از نوع early stopping. که اگر val_acc در تعداد ایپاک معین تغییری نکرد، آموزش مدل را متوقف کند تا اورفیت نشود.
- کاهش پیچیدگی مدل (مثلا در اینجا کاهش سایز لایه embedding)
- Feature selection مناسب

- تغییر نرخ یادگیری (learning rate)
- تغییر activation function

ما در مدل های ارائه شده از پیش پردازش،dropout ، callback ، کاهش سایز embedding استفاده کردیم.

سوال ۱۵)

افزایش تعداد لایه ها مشکل vanishing gradient را حل نمی کند. کاهش تعداد لایه ها موثر است.

با کاهش تعداد لایه‌ها در شبکه، برخی از پیچیدگی‌های مدل‌های خود را کنار می‌گذاریم، زیرا داشتن لایه‌های بیشتر باعث می‌شود شبکه‌ها توانایی بیشتری برای نمایش map های پیچیده داشته باشند.
