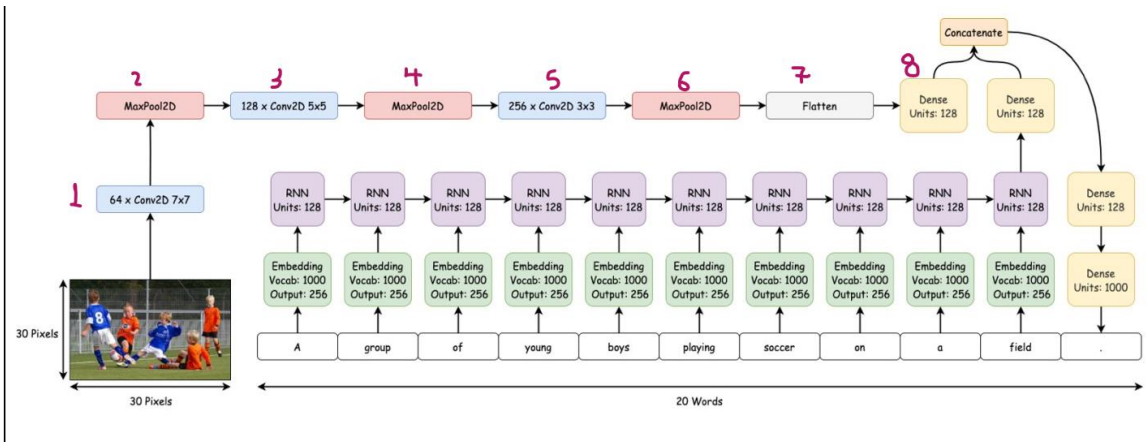
به نام خدا

تمرین یازدهم یادگیری عمیق

غزل زمانی‌نژاد

۹۷۵۲۲۱۶۶

. .1



حسابات كبش Convolution :

$$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} \right\rfloor + 1$$

1 لايه : $n_{out} = 30 - 7 + 1 = 24 \rightarrow (24, 24, 64)$

2 لايه : $n_{out} = 24 - 2 + 1 = 23 \rightarrow (23, 23, 64)$

3 لايه : $n_{out} = 23 - 5 + 1 = 19 \rightarrow (19, 19, 128)$

4 لايه : $n_{out} = 19 - 2 + 1 = 18 \rightarrow (18, 18, 128)$

5 لايه : $n_{out} = 18 - 3 + 1 = 16 \rightarrow (16, 16, 256)$

6 لايه : $n_{out} = 16 - 2 + 1 = 15 \rightarrow (15, 15, 256)$

7 لايه : $15 \times 15 \times 256 = 57600 \rightarrow (57600,)$

8 لايه : $\rightarrow (128)$

حسابات لايه embedding :

Embedding (1000, 256) $\xrightarrow[\text{output}]{}$ (1000, 20, 256)

seq_len embed_dim

محاسبات لایه RNN

چون return seq = false است تنها بعدی از RNN خروجی دارد

↳ (128,)

Dense → output (128,)

Concat → output (256,)

Dense → output (128,)

Dense → output (100)

تعداد پارامترهای کانولوشن ، چون ورودی 3 کاناله است

لایه 1 $[(7 \times 7 \times 3) + 1] \times 64 = 148 \times 64 = 9472$

لایه 2 → پارامتر trainable ندارد → n = 0

لایه 3 $[(5 \times 5 \times 64) + 1] \times 128 = 204,928$

لایه 4 n = 0

لایه 5 $[(3 \times 3 \times 128) + 1] \times 256 = 1153 \times 256 = 294,912$

لایه 6 n = 0

لایه 7 n = 0

لایه 8 $(57600 + 1) \times 128 = 7,372,928$

embedding → $n = 1000 \times 64 = 64{,}000$

simpleRNN(128, input=(20, 256))

$128 \times 128 + 256 \times 128 + 128 = 49{,}280$

2. ابتدا دیتاست را دانلود میکنیم

```
from nltk.corpus import reuters
print(len(reuters.fileids('crude')))
print(reuters.categories())
```

```
578
['acq', 'alum', 'barley', 'bop', 'carcass
```

سپس برای دسته بندی crude جملات اخبار را می سازیم.

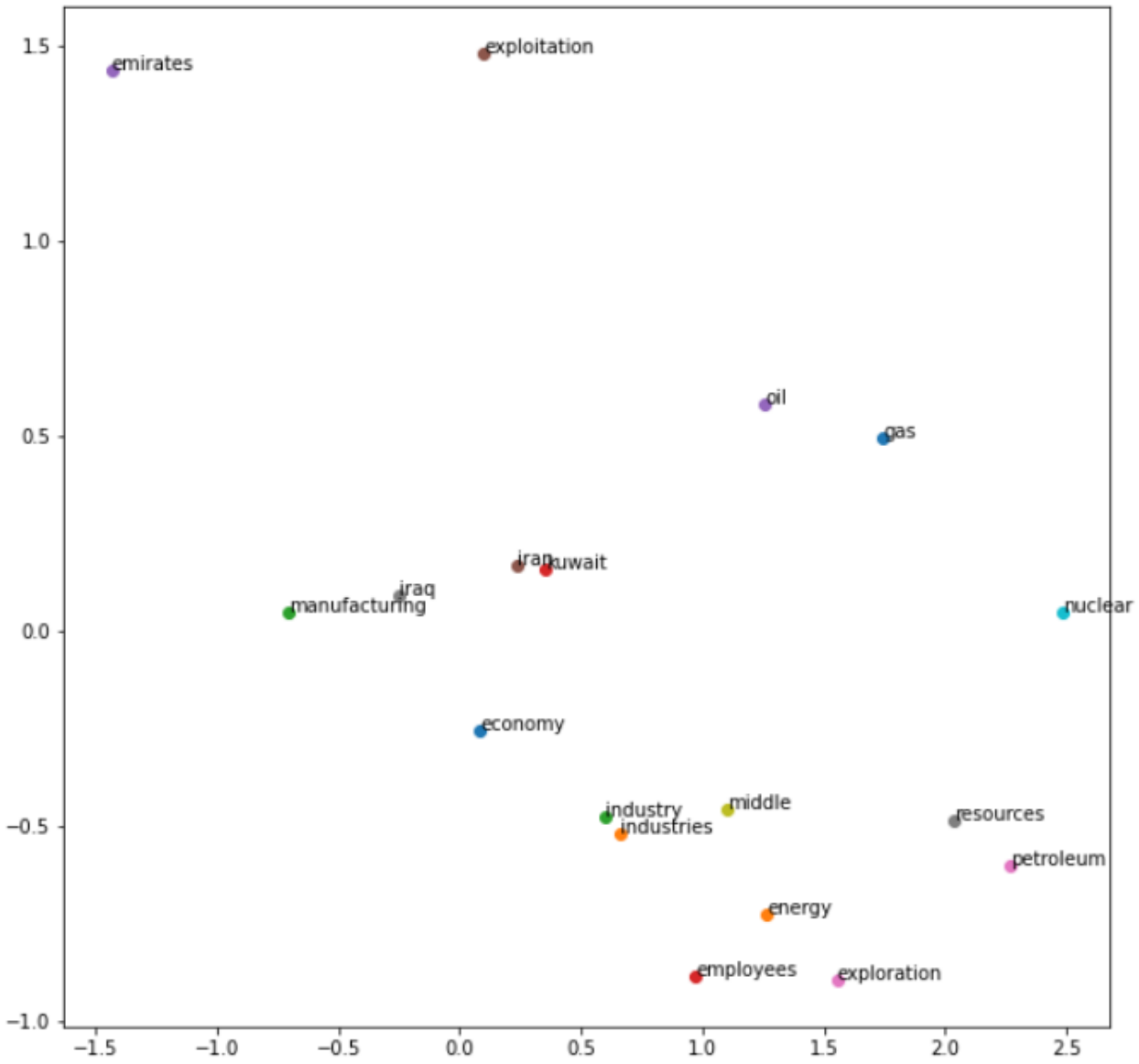
```
data = reuters.fileids('crude')[:100]
news = []
for d in data:
  words = reuters.words(d)
  news.append(' '.join(words))

print(news[10])
```

از کدهای گیت هاب برای یافتن embeddingها استفاده میکنیم. نتیجه آموزش مدل در ایپوک های پایانی:

```
Epoch 993/1000
221/221 [==============================] - 2s 9ms/step - loss: 6.1763
Epoch 994/1000
221/221 [==============================] - 2s 9ms/step - loss: 6.1762
Epoch 995/1000
221/221 [==============================] - 2s 9ms/step - loss: 6.1762
Epoch 996/1000
221/221 [==============================] - 2s 9ms/step - loss: 6.1762
Epoch 997/1000
221/221 [==============================] - 2s 9ms/step - loss: 6.1762
Epoch 998/1000
221/221 [==============================] - 2s 9ms/step - loss: 6.1763
Epoch 999/1000
221/221 [==============================] - 2s 8ms/step - loss: 6.1761
Epoch 1000/1000
221/221 [==============================] - 2s 8ms/step - loss: 6.1761
```

نمودار کلمات خواسته شده را چاپ میکنیم.

```
# Ploting the embeddings
plt.figure(figsize=(10, 10))
plt_words = ['gas', 'energy', 'industry', 'kuwait', 'oil', 'iran', 'petroleum', 'iraq',
             'middle', 'nuclear', 'economy', 'industries', 'manufacturing', 'employees',
             'emirates', 'exploitation', 'exploration', 'resources']
# for word in list(unique_word_dict.keys()):
for word in plt_words:
    coord = embedding_dict.get(word)
    plt.scatter(coord[0], coord[1])
    plt.annotate(word, (coord[0], coord[1]))
```



بررسی: همانطور که انتظار میرفت کلمات مشابه در این صفحه متخصات در نزدیکی یکدیگر قرار گرفته اند. مثلا کلمات ایران، کویت و عراق فاصله کمی از هم دارند و این نشان میدهد مدل توانسته به خوبی یاد بگیرد که این 3 لغت در یک دسته بندی (کشور) به یکدیگر مربوط هستند. همینطور کلمات oil و gas، industry و industries و ..

برای چند کلمه با استفاده از متد find_similar کلمات مشابه با آن را می یابیم.

```
1 find_similar('iran', embedding_dict)
```

```
[('communication', 0.05752556),
 ('talking', 0.06276985),
 ('supply', 0.06379738),
 ('supplies', 0.068744384),
 ('increased', 0.07603855),
 ('producer', 0.08262307),
 ('agreement', 0.08572599),
 ('annual', 0.09117655),
 ('settlement', 0.10088544),
 ('inch', 0.1010083)]
```

```
1 find_similar('aware', embedding_dict)
```

```
[('agrement', 0.026115535),
 ('something', 0.037052095),
 ('algerians', 0.03987449),
 ('ships', 0.058817934),
 ('follow', 0.10351497),
 ('arrangement', 0.10861039),
 ('condtions', 0.115833566),
 ('re', 0.12418954),
 ('leblanc', 0.13774122),
 ('patrolling', 0.1491283)]
```

```
1 find_similar('jungle', embedding_dict)
```

```
[('heart', 0.22345117),
 ('reference', 0.26099488),
 ('annum', 0.3571673),
 ('oilfields', 0.36861652),
 ('lighter', 0.3841773),
 ('collapsed', 0.43634808),
 ('nine', 0.4437281),
 ('marginally', 0.45145908),
 ('faces', 0.48300773),
 ('metric', 0.48596492)]
```

```
1 find_similar('justice', embedding_dict)
```

```
[('backed', 0.032110613),
 ('consuming', 0.07040953),
 ('for', 0.08582186),
 ('proration', 0.10396621),
 ('india', 0.104285814),
 ('reduced', 0.11137408),
 ('adherence', 0.11648192),
 ('of', 0.12503901),
 ('while', 0.12536553),
 ('demand', 0.13178091)]
```

بررسی: مشاهده میشود که مدل با توجه به اخباری که بر روی آن آموزش دیده، تعدادی کلمه مشابه برای لغات پیدا کرده است. مثلا برای کلمه iran، کلمات agreement, talking, communication و .. بیشترین شباهت را با آن داشته اند. چون دیتاست مربوط به اخبار بوده، مدل شبیه ترین لغات را به ایران مربوط به ارتباط، سخنرانی و توافق دانسته است. برای سایر کلمات نیز مشابه با context ای که در آن آموزش دیده کلمات مشابه را یافته است.

پاسخ سوالات تئوری:

- **پیش پردازش ها در متد clean_text انجام شده است. url و html elementها از داخل رشته ها حذف شدند. علائم نقطه گذاری از رشته ها حذف شدند. تمامی کاراکترها را به lower case تبدیل کردیم تا تمامی لغات یکدست باشند. Stop words کلماتی هستند که در زبان انگلیسی بسیار پرتکرار هستند و در نتیجه در فرآیند یادگیری به مدل کمکی نمی کنند. به همین دلیل آنها را از رشته ها حذف میکنیم. در نهایت whitespaceها را نیز حذف میکنیم.**
- **در لیست کلمات منحصر به فرد این کلاس iterate کردیم. برای x داده آموزشی، value آن را در دیکشنری با کلید کلمه بدست آوردیم و به صورت one hot انکود کردیم. برای برچسب داده آموزشی y، از context استفاده کردیم. کلمه بعدی را در نظر گرفتیم و در دیکشنری value آن را پیدا کردیم. و آن را و به صورت one hot انکود کردیم. تاپل (x,y) به عنوان یک نمونه آموزشی در نظر میگیریم.**
- **مقدار window مشخص میکند چه تعدادی از کلمات را برای context یک کلمه در نظر بگیریم. هرچه بیشتر باشد کلمات همسایه ای که برای کلمه مشخص در نظر میگیریم بیشتر می شود. در نتیجه embeddingها دقیقتر میشوند. اما هزینه محاسباتی بیشتر می شود.**
- **برای ساختن embedding نهایی، ابتدا شبکه را در 1000 ایپوک آموزش دادیم. سپس از وزن های شبکه استفاده کردیم و وزن های آموخته شده متناظر با آن کلمه را برای بردار embedding آن درنظر گرفتیم.**

3. ابتدا دیتاست را دانلود میکنیم. آن را شافل میکنیم تا ترتیب داده بهم بریزد.

```
from tensorflow.keras.datasets import mnist
from matplotlib import pyplot as plt

(orig_x_train, orig_y_train), (orig_x_test, orig_y_test) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
```

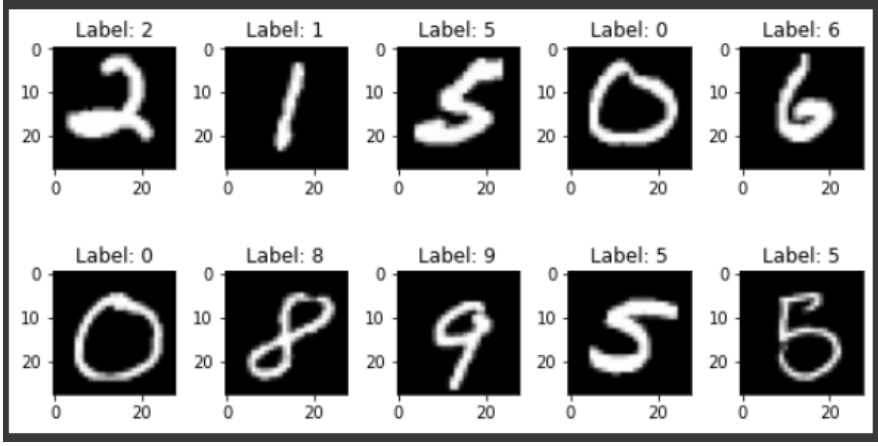
```
from sklearn.utils import shuffle
orig_x_train, orig_y_train = shuffle(orig_x_train, orig_y_train, random_state=2)
orig_x_test, orig_y_test = shuffle(orig_x_test, orig_y_test, random_state=5)
```

```
print("train input shape", orig_x_train.shape)
print("train label shape", orig_y_train.shape)
print("test input shape", orig_x_test.shape)
print("test label shape", orig_y_test.shape)
```

```
train input shape (60000, 28, 28)
train label shape (60000,)
test input shape (10000, 28, 28)
test label shape (10000,)
```

سپس با استفاده از کتابخانه pyplot 10 تصویر نخست به همراه لیبل را در یک حلقه چاپ میکنیم.

```
num_row = 2
num_col = 5

num = num_row*num_col
images = orig_x_train[:num]
labels = orig_y_train[:num]

fig, axes = plt.subplots(num_row, num_col, figsize=(1.5*num_col,2*num_row))
for i in range(num_row*num_col):
    ax = axes[i//num_col, i%num_col]
    ax.imshow(images[i], cmap='gray')
    ax.set_title('Label: {}'.format(labels[i]))
plt.tight_layout()
plt.show()
```



ابعاد ورودی را به گونه ای تغییر میدهیم که بتواند به عنوان وروردی کانولوشن قرار بگیرد. داده ها را نرمالایز میکنیم و لیبل ها را به صورت one hot، انکود میکنیم.

```
import numpy as np

x_train = np.expand_dims(orig_x_train, 3)
x_test = np.expand_dims(orig_x_test, 3)
print("train input after expand", x_train.shape)
print("test input after expand", x_test.shape)
```

```
train input after expand (60000, 28, 28, 1)
test input after expand (10000, 28, 28, 1)
```

```
from tensorflow.keras.utils import to_categorical

x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255
print(x_train.shape)
print(x_test.shape)

y_train = to_categorical(orig_y_train, num_classes=10)
y_test = to_categorical(orig_y_test, num_classes=10)
print(y_train.shape)
print(y_test.shape)
```

```
(60000, 28, 28, 1)
(10000, 28, 28, 1)
(60000, 10)
(10000, 10)
```

مدل را مطابق چیزی که خواسته شده تعریف میکنیم:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import CategoricalCrossentropy

model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=(28, 28, 1)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

ابتدا آن را کامپایل میکنیم و سپس در 15 ایپوک مدل را آموزش میدهیم.

```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit(x_train, y_train, epochs=15, batch_size=64, validation_data=(x_test, y_test), verbose=0)
```

نتیجه آموزش برای ایپوک های پایانی مطابق زیر است:

```
****************************************
epoch 11
loss: 0.007857410237193108
accuracy: 0.9975333213806152
val_loss: 0.029060961678624153
val_accuracy: 0.9926000237464905
****************************************
epoch 12
loss: 0.00667792372405529
accuracy: 0.9975000023841858
val_loss: 0.029362255707383156
val_accuracy: 0.9918000102043152
****************************************
epoch 13
loss: 0.007952702231705189
accuracy: 0.9974333047866821
val_loss: 0.0324624665081501
val_accuracy: 0.9908000230789185
****************************************
epoch 14
loss: 0.006294935010373592
accuracy: 0.9979666471481323
val_loss: 0.02947588823735714
val_accuracy: 0.9926000237464905
****************************************
epoch 15
loss: 0.005352174863219261
accuracy: 0.998199999332428
val_loss: 0.033827703446149826
val_accuracy: 0.9915000200271606
****************************************
```

سپس به کمک یک حلقه آخرین لایه کانولوشنی را در یک متغیر ذخیره میکنیم.

```
i = 0
last_conv = None
for l in model.layers:
  if i == 4:
    last_conv = l
    break
  i += 1

print(last_conv.name)
```

```
conv2d_2
```

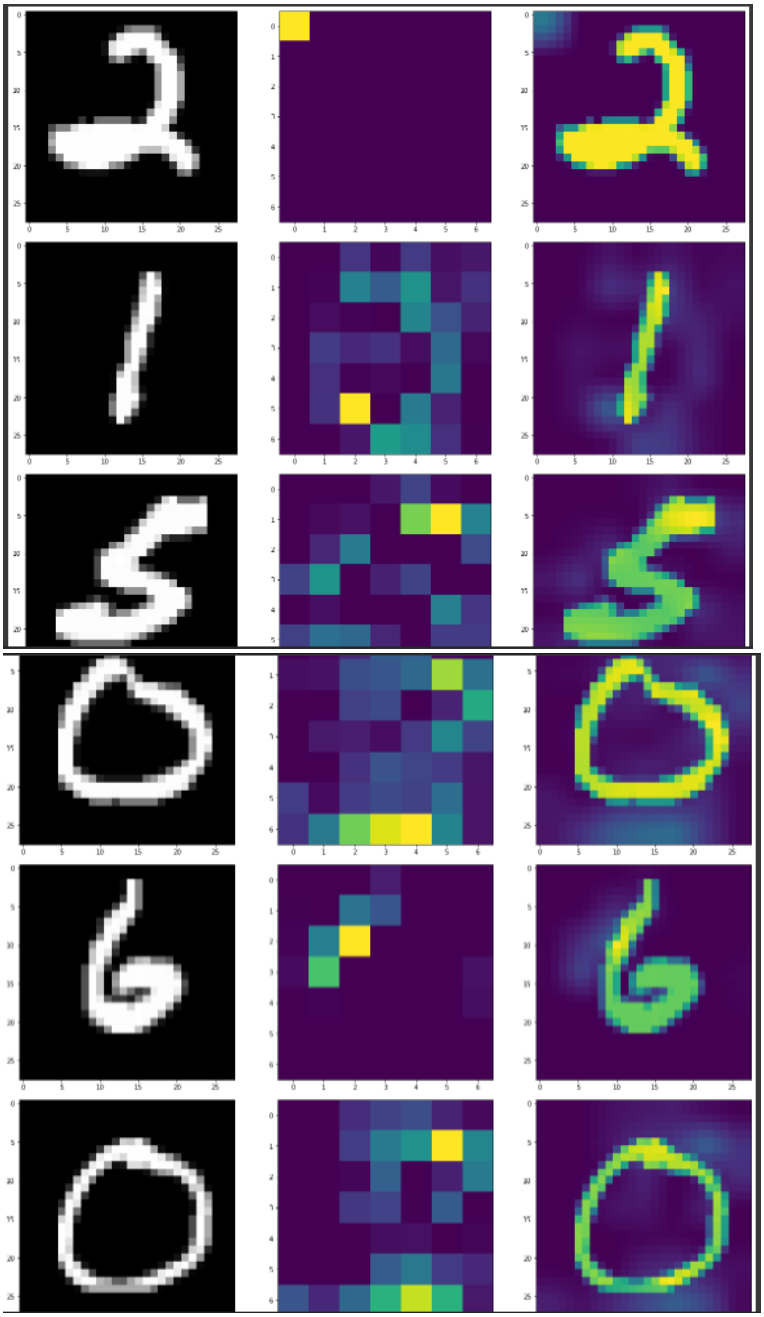
سپس برای 10 تصویر نخست، نقشه حرارتی آنها را در یک حلقه محاسبه میکنیم. برای این کار مشابه اسلاید عمل میکنیم.

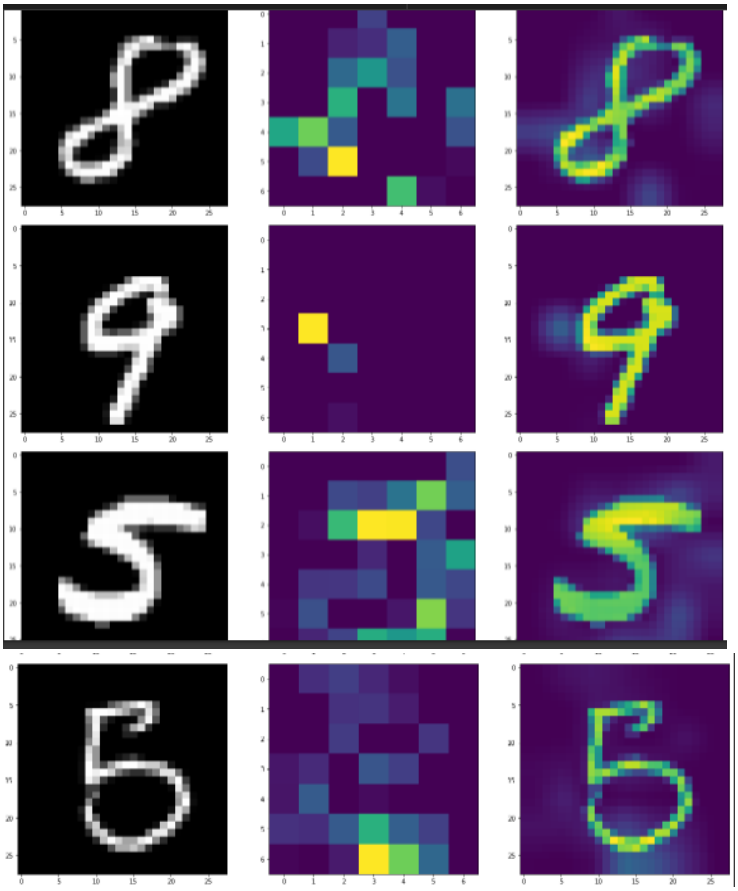
```
heatmap_model = models.Model([model.inputs], [last_conv.output, model.output])

for i in range(10):
  img = orig_x_train[i]
  sample = x_train[i]
  sample = np.expand_dims(sample, 0)

  # compute heatmap
  with tf.GradientTape() as gtape:
      conv_output, predictions = heatmap_model(sample)
      loss = predictions[:, np.argmax(predictions[0])]
      grads = gtape.gradient(loss, conv_output)
      pooled_grads = K.mean(grads, axis=(0, 1, 2))

  heatmap = tf.reduce_mean(tf.multiply(pooled_grads, conv_output), axis=-1)
  heatmap = np.maximum(heatmap, 0)
  max_heat = np.max(heatmap)
  if max_heat == 0:
      max_heat = 1e-10
  heatmap /= max_heat
  heatmap = np.squeeze(heatmap)

  # add heatmap to image
  heat = cv2.resize(heatmap, (img.shape[1], img.shape[0]))
  heat = np.uint8(255*heat)
  superimposed = heat * 0.4 + img

  # plot
  fig, ax = plt.subplots(1, 3)
  fig.set_size_inches(20,20)

  ax[0].imshow(img, cmap="gray")
  ax[1].imshow(heatmap)
  ax[2].imshow(superimposed)
```

ابتدا تصویر مربوطه را پیدا میکنیم. سپس تخمین مدل هیت مپ را برای آن می یابیم. برای محاسبه میزان ضرر از تابع gradient استفاده میکنیم. سپس با استفاده از backend موجود در کراس میانگین شدت گرادیان در یک کانال برای یک feature map را حساب میکنیم. آن را در خروجی ضرب میکنیم و دوباره میانگین میگیریم. بر روی نقشه حرارتی post-process انجام میدهیم. یعنی مقادیر منفی را از بین میبریم و آن را نرمالایز میکنیم. آن را squeeze میکنیم تا تعداد بعدهای آن مطابق تصویر شود.

در پایان ابعاد نقشه حرارتی را مطابق ابعاد تصویر قرار میدهیم. اثر آن را مقداری کم میکنیم (ضربدر 0.4) و با تصویر جمع میکنیم.

نتایج بدست آمده را چاپ میکنیم:





تحلیل نتایج: شبکه برای اینکه بتواند هر تصویر را به درستی برچسب بزند، باید به بخش هایی از تصویر که بیشتر به پیش بینی کمک میکنند، بیشتر توجه کند. برای اینکار با استفاده از الگوریتم grad cam نقشه ویژگی را چاپ کردیم تا ببینیم لایه آخر کانولوشنی برای 10 تصویر تصادفی به چه نقاطی بیشتر توجه میکند. و همانطور که انتظار میرفت شبکه به نواحی درستی توجه کرده است. مثلا در تصویر آخر نقشه ویژگی به پیکسل های سفیدی که در تصویر موجود هستند و به پیش بینی کمک میکنند بیشتر توجه کرده است (پیکسل های آبی کمرنگ و زرد)

منابع

https://stackoverflow.com/questions/58322147/how-to-generate-cnn-heatmaps-using-built-in-keras-in-tf2-0-tf-keras