گزارش تمرین یازدهم بینایی ماشین

نام تهیه‌کننده: ملیکا نوبختیان

شماره دانشجویی: ۹۷۵۲۲۰۹۴

نسخه: ۱

۱- سوال اول

الف)

در لایه‌های fully connected مقدار یک نورون در یک لایه به تمام مقادیر لایه قبل وابسته است در حالی در بسیاری از مواردی ما نیازی نداریم تا اطلاعات تمام پیکسل‌های یک تصویر را داشته باشیم، بلکه اطلاعات تنها یک قسمت محدود و محلی کافی و حتی مناسب‌تر است. فیلترهای لایه conv نیز تنها این اطلاعات محلی را در هر قسمت مورد بررسی قرار می‌دهند و با share کردن وزن‌ها علاوه بر داشتن پارامترهای کمتر، عملکرد بهتری نیز دارند.

ب)

$$W = \frac{(W - F + 2P)}{S} + 1 \qquad 16 = \frac{16 - 5 + 2P}{1} + 1$$

$$15 = 11 + 2P \qquad 2P = 4 \qquad \boxed{P = 2}$$

تعداد پارامترهای لایه convolational برای هر فیلتر $= ((m * n * d) + 1) * k$

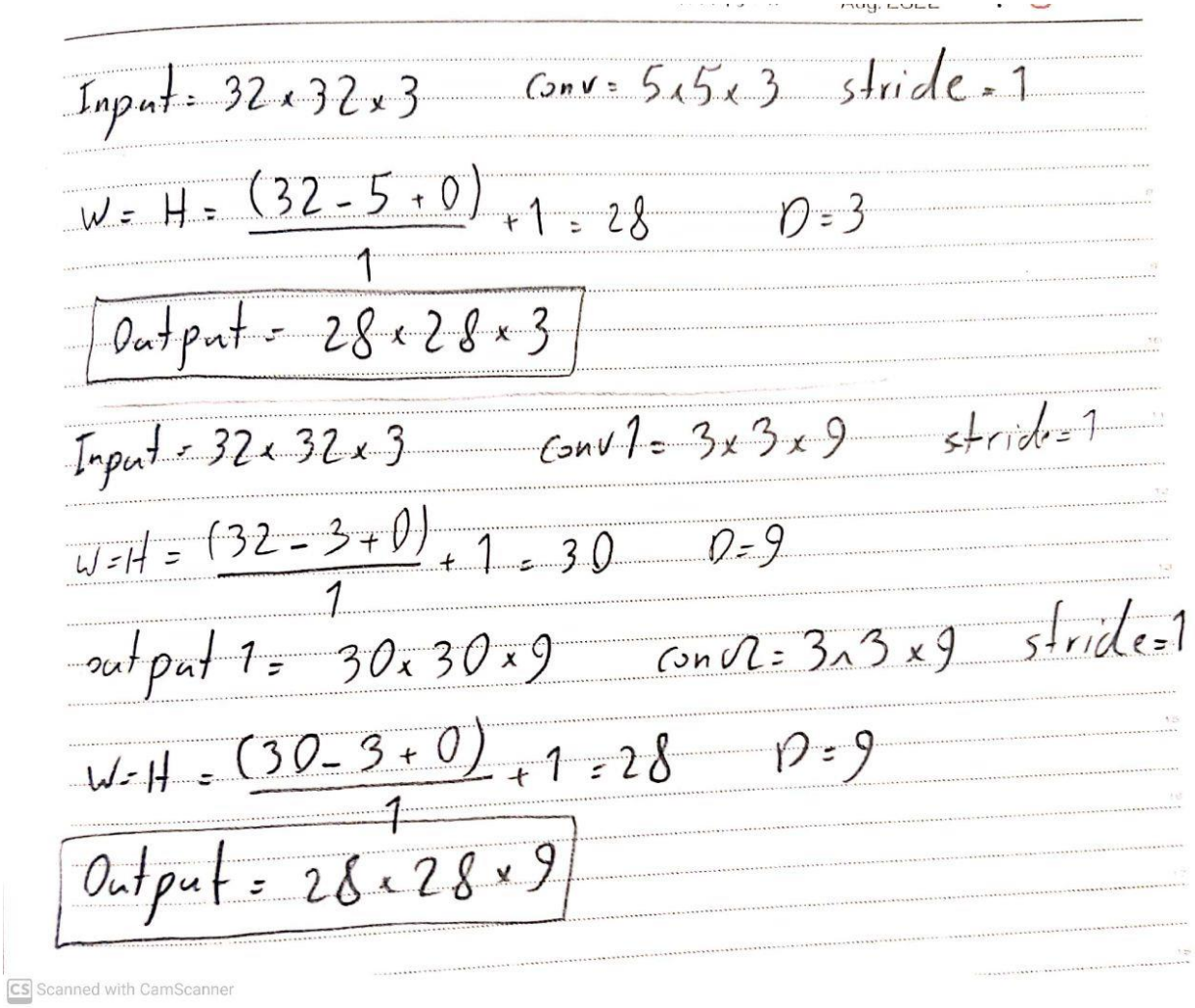
m = width n = height d: تعداد فیلترهای تصویری که در این لایه داریم

1 بابت bias k = تعداد فیلترهای این لایه

$$= ((5 \times 5 \times 5) + 1) * 16 = \boxed{2016}$$

پارامتر

ج)



ت)

Max pooling: این نوع از pooling رایج‌ترین نوع است و در کاربردهای دسته‌بندی تصاویر و به طور کلی computer vision کاربرد بسیاری دارد. این pooling شاخص‌ترین ویژگی را به ما بازمی‌گرداند و مهم‌ترین مشخصه و مزیت آن translation invariance بودن آن است، یعنی نسبت به کارهایی مانند rotation, shift و scale تغییری ندارد. این خاصیت زمانی به کار می‌آید که ما می‌خواهیم یک شی در تصویر را فارغ از جایی که وجود دارد شناسایی کنیم.

Average pooling: به طور معمول max pooling بیشتر از average pooling کاربرد دارد اما اگر موقعیت یک جسم در تصویر برای ما اهمیت داشته باشد، average pooling گزینه مناسب‌تری برای ما خواهد بود.

Global average pooling: این لایه در ابتدا خاصیت مهم لایه‌های global pooling را دارا است که پارامترها را هنگامی که در نهایت می‌خواهیم تصویر را به یک لایه fully connected بدهیم کاهش می‌دهد. هم چنین این نوع

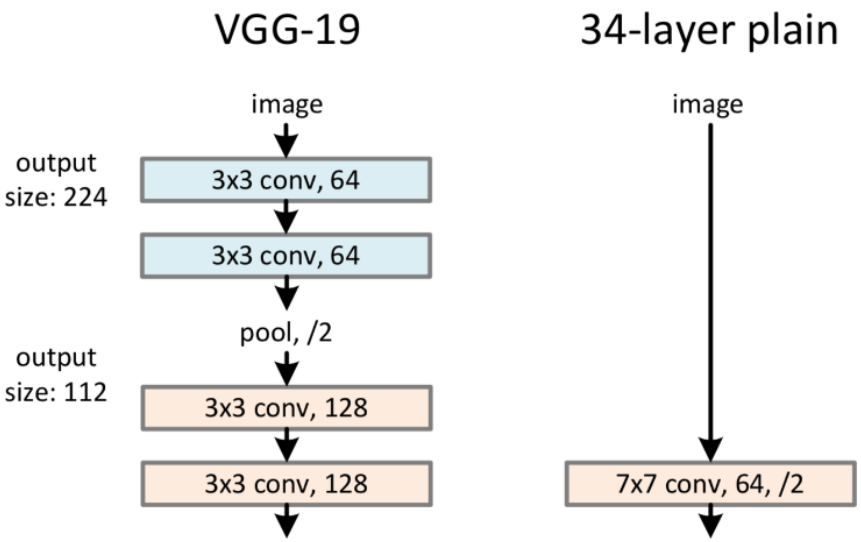
از pooling خاصیت object localization را نیز دارد، یعنی می‌تواند به ما بگوید که یک جسم در کدام قسمت از تصویر قرار دارد که به خاطر average بودن آن است.

ث)

ایده VGG برای بهبود مدل‌های کانولوشنی استفاده از فیلترهای کوچک‌تر و لایه‌های بیشتر بود تا با وجود استفاده از یک شبکه عمیق‌تر، پارامترهای لازم نسبت به AlexNet کاهش یابد. در این شبکه لایه‌ها از ۸ به ۱۶ و ۱۹ لایه افزایش یافته‌است. در این شبکه تعداد فیلترها به تدریج افزایش می‌یابد. تمام فیلترها در این شبکه 3*3 هستند و هر دو لایه conv با یه لایه pooling همراه می‌شود.

ResNet با ۱۵۲ لایه انقلابی در عمق شبکه‌های کانولوشنی به وجود آورد. اما به طور معمول وقتی تعداد لایه‌ها در یک شبکه زیاد می‌شد، آموزش آن با مشکل مواجه می‌شد چون بهینه‌سازی آن دشوار بود و با مشکلاتی نظیر vanishing/exploding gradient مواجه می‌شدیم. اما ایده ResNet برای هر این مشکل این بود که شبکه به جای آموختن نگاشت مطلوب، باقی‌مانده آن را یاد بگیرند. به جز لایه اول هر بلوک باقی‌مانده دارای فیلترهای ۳*۳ است. به طور دوره‌ای تعداد فیلترها دو برابر شده و رزولوشن مکانی نصف می‌شود.

در مورد مقایسه سرعت ResNet و VGG علاوه بر تعداد پارامترها باید به موضوع دیگری به نام FLOP بپردازیم که تاثیر زیادی در سرعت دارد. FLOP به معنای floating operations per second است و هر چه تعداد این عملیات‌ها بیشتر باشد، سرعت مدل نیز کمتر خواهد شد. برای مثال اگر یک فیلتر ۳*۳ را به یک تصویر اعمال کنیم، به ازای هر بار اعمال کردن به ۹ بار ضرب کردن و ۸ بار جمع کردن نیاز خواهیم داشت که در مجموع ۱۷ عملیات خواهد بود. هرچه ابعاد تصویر بیشتر باشد، تعداد این عملیات‌ها نیز بیشتر خواهد شد. تفاوت VGG و Resnet در همین جا و لایه‌های اولیه سرچشمه می‌گیرد:



همان طور که در تصویر مشخص است VGG در ابتدا با فیلترهای کوچک‌تر روی تصویر سر و کار دارد و به همین دلیل ابعاد تصاویر در این لایه‌ها بزرگ خواهد بود و نیاز به تعداد زیادی FLOP خواهیم داشت. در حالی که ResNet با اعمال یک فیلتر بزرگ در همان ابتدا اندازه تصویر را به شکل خوبی کاهش می‌دهد و به این ترتیب با

توجه به وجود چند لایه conv اولیه موجود در ابتدای مدل VGG و فیلترهای کوچک موجود FLOP کمتر و سرعت بیشتری خواهد داشت.

۲- سوال دوم

مدل fully connected را به شکل زیر تعریف کردم:

```
fc_model.add(keras.layers.Flatten(input_shape=x_train[0].shape))
fc_model.add(keras.layers.Dense(units=128, activation='relu'))
fc_model.add(keras.layers.Dense(units=512, activation='relu'))
fc_model.add(keras.layers.Dense(units=256, activation='relu'))
fc_model.add(keras.layers.Dense(units=num_classes, activation='softmax'))
```

در نهایت اطلاعات مدل به شکل زیر خواهد شد:

```
Model: "sequential_14"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_14 (Flatten)        (None, 3072)              0

 dense_36 (Dense)            (None, 128)               393344

 dense_37 (Dense)            (None, 512)               66048

 dense_38 (Dense)            (None, 256)               131328

 dense_39 (Dense)            (None, 10)                2570

=================================================================
Total params: 593,290
Trainable params: 593,290
Non-trainable params: 0
```

مدل کانولوشنی را نیز به شکل زیر تعریف کردم:

```
Conv_model.add(keras.layers.Conv2D(filters=128, kernel_size=(5, 5), activa
tion='relu'))
Conv_model.add(keras.layers.Conv2D(filters=64, kernel_size=(3, 3), activat
ion='relu'))
Conv_model.add(keras.layers.Conv2D(filters=32, kernel_size=(3, 3), activat
ion='relu'))
Conv_model.add(keras.layers.Flatten())
Conv_model.add(keras.layers.Dense(units=num_classes, activation='softmax')
)
```

و اطلاعات مدل نیز به شرح زیر است:

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_6 (Conv2D)           (None, 28, 28, 128)       9728

 conv2d_7 (Conv2D)           (None, 26, 26, 64)        73792

 conv2d_8 (Conv2D)           (None, 24, 24, 32)        18464

 flatten_4 (Flatten)         (None, 18432)             0

 dense_5 (Dense)             (None, 10)                184330

=================================================================
Total params: 286,314
Trainable params: 286,314
Non-trainable params: 0
```

همان طور که در تصاویر مشخص است مدل کانولوشنی تقریبا نصف مدل fully connected پارامتر دارد. برای لایه‌های میانی هر دو مدل از تابع فعال‌سازی relu استفاده کردم و برای لایه نهایی هر دو مدل از softmax استفاده کردم تا احتمال کلاس‌های مختلف به دست بیاید.

نتایج آموزش مدل fully connected و هم چنین نتایج روی داده‌های تست به شکل زیر است:

```
Fully Connected Model
Epoch 1/5
1563/1563 [==============================] - 6s 4ms/step - loss: 1.9515 - accuracy: 0.2801
Epoch 2/5
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8281 - accuracy: 0.3321
Epoch 3/5
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7846 - accuracy: 0.3524
Epoch 4/5
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7529 - accuracy: 0.3638
Epoch 5/5
1563/1563 [==============================] - 5s 3ms/step - loss: 1.7349 - accuracy: 0.3694


Loss and Accuracy on Test set :
313/313 [==============================] - 1s 3ms/step - loss: 1.8096 - accuracy: 0.3394
```

و نتایج train و test مدل کانولوشنی نیز به شکل زیر است:

```
Convolutional Model
Epoch 1/5
1563/1563 [==============================] - 11s 7ms/step - loss: 0.9662 - accuracy: 0.6613
Epoch 2/5
1563/1563 [==============================] - 10s 7ms/step - loss: 0.8767 - accuracy: 0.6932
Epoch 3/5
1563/1563 [==============================] - 10s 7ms/step - loss: 0.7914 - accuracy: 0.7236
Epoch 4/5
1563/1563 [==============================] - 11s 7ms/step - loss: 0.7166 - accuracy: 0.7504
Epoch 5/5
1563/1563 [==============================] - 10s 6ms/step - loss: 0.6469 - accuracy: 0.7731


Loss and Accuracy on Test set :
313/313 [==============================] - 1s 4ms/step - loss: 1.6625 - accuracy: 0.5403
```

در مدل fc میزان تابع ضرر روی داده تست برابر 1.80 و میزان دقت برابر ۳۳ درصد است. در مدل conv مقدار تابع ضرر برابر 1.66 و دقت برابر ۵۴ درصد است.

در مورد خطا و دقت باید این نکته را بدانیم که لزوما کم بودن خطا به زیاد بودن دقت منجر نمی‌شود زیرا روابط این دو پارامتر به این شکل نیستند که دقیقا کم شدن یکی زیاد شدن دیگری را نتیجه دهد. تابع ضرر ما در اینجا categorical crossentropy است و به نسبت اینکه احتمالی که پیش‌بینی می‌کنیم به برچسب اصلی نزدیک‌تر باشد و هم چنین درصد احتمال برای برچسب‌های غیرمرتبط کمتر باشد، میزان تابع ضرر کمتر خواهد شد. دقت اما به این مربوط است که چقدر توانسته‌ایم درست برچسب داده‌ها را پیش‌بینی کنیم. هر چند کاهش تابع ضرر در بیشتر موارد دقت بالا را به همراه خواهد داشت اما ممکن است حالتی پیش بیاید که مقدار تابع ضرر کم باشد اما مقدار دقت نیز پایین باشد. این مورد زمانی می‌تواند رخ دهد که هر چند میزان احتمالی که برای برچسب درست به دست آورده‌ایم زیاد است اما برچسب دیگری وجود دارد که احتمال بالاتری دارد و به عنوان برچسب پیش‌بینی شده انتخاب می‌شود. در این حالت تابع ضرر کم به دقت بالا منجر نخواهد شد.

در مدل FC مدت زمان اجرای هر epoch حدودا ۵ ثانیه و در مدل Conv حدودا ۱۰ ثانیه است.

هر چند که به نظر می‌رسد مدل Conv با پارامترهای کمتر باید زمان کمتری برای آموزش نیاز داشته باشد اما با توجه به زمان هر epoch برای هر مدل عکس این قضیه را مشاهده می‌کنیم، مدل Conv با پارامترهای کمتر نیازمند زمان بیشتری برای آموزش بوده‌است! این موضوع به این دلیل است که پیچیدگی محاسبات در شبکه Conv بیشتر از FC است. هر چند تعداد پارامترهای مدل FC بیشتر است اما به نسبت مدل Conv به تعداد ضرب‌های کمتری نیاز دارد که سرعت آن را افزایش می‌دهد.

۳- سوال سوم

در ابتدا تابع resize را به شکل زیر تعریف می‌کنیم:

```
def resize_img(img, desired_size = 224):
    # write your code here
    channels = 3
    old_size = img.shape
    ratio = float(desired_size) / max(old_size)
    new_size = (int(old_size[1] * ratio), int(old_size[0] * ratio))
    img = cv2.resize(img, new_size)
    diff_w = desired_size - new_size[0]
    diff_h = desired_size - new_size[1]
    top, bottom = diff_h//2, diff_h-(diff_h//2)
    left, right = diff_w//2, diff_w-(diff_w//2)
    new_img = np.zeros((desired_size, desired_size, channels), dtype=img.d
type)
    for i in range(channels):
      ch = img[:, :, i]
      new_ch = np.pad(ch, ((top, bottom),(left, right)), 'constant')
      new_img[:, :, i] = new_ch
```

```
    return new_img
```

در گام اول لازم است تصویری که داریم را با توجه به ابعادش و هم چنین ابعاد خروجی موردنظرمان scale کنیم. برای این کار نسبت تبدیل را با تقسیم کردن ابعاد خروجی به بزرگ ترین بعد تصویر کنونی به دست می‌آوریم و سپس با ضرب کردن ابعاد تصویر اصلی در این نسبت آن را scale می‌کنیم. حالا لازم است بقیه تصویر را pad کنیم تا به سایز موردنظر برسانیم. پیکسل‌های pad شده را با مقدار صفر پر می‌کنیم تا در نهایت به تصویر پردازش‌شده برسیم.

در ابتدا مدل ResNet را با وزن‌های رندوم به شکل زیر تعریف می‌کنیم:

```
resnet = tf.keras.models.Sequential()
# Write your code here
resnet.add(keras.layers.Input(shape=(224, 224, 3)))
resnet_50 = tf.keras.applications.ResNet50(include_top=True, weights=None, classes=24)
resnet.add(resnet_50)
resnet.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50 (Functional)       (None, 24)                23636888

=================================================================
Total params: 23,636,888
Trainable params: 23,583,768
Non-trainable params: 53,120
_________________________________________________________________
```

نتیجه آموزش مدل در 5 epoch به شکل زیر است:

```
Epoch 1/5
65/65 [==============================] - 84s 1s/step - loss: 0.7884 - acc: 0.7864
Epoch 2/5
65/65 [==============================] - 65s 990ms/step - loss: 0.2194 - acc: 0.9270
Epoch 3/5
65/65 [==============================] - 63s 964ms/step - loss: 0.0466 - acc: 0.9856
Epoch 4/5
65/65 [==============================] - 63s 963ms/step - loss: 0.0484 - acc: 0.9849
Epoch 5/5
65/65 [==============================] - 63s 962ms/step - loss: 0.2132 - acc: 0.9350
<keras.callbacks.History at 0x7f093033dcd0>
```

حالا مدل ResNet را با وزن‌های imagenet به شکل زیر تعریف می‌کنیم و وزن‌های آن را فریز می‌کنیم:

```
fine_tune_resnet = tf.keras.models.Sequential()
# write your code here
fine_tune_resnet.add(keras.layers.Input(shape=(224, 224, 3)))
resnet_50 = tf.keras.applications.ResNet50(include_top=False, weights='imagenet')
resnet_50.trainable = False
fine_tune_resnet.add(resnet_50)
fine_tune_resnet.add(keras.layers.Flatten())
fine_tune_resnet.add(keras.layers.Dense(units=24, activation='softmax'))
fine_tune_resnet.summary()

```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50 (Functional)       (None, None, None, 2048)  23587712

 flatten_1 (Flatten)         (None, 100352)            0

 dense_1 (Dense)             (None, 24)                2408472

=================================================================
Total params: 25,996,184
Trainable params: 2,408,472
Non-trainable params: 23,587,712
_________________________________________________________________
```

نتایج آموزش این مدل به شکل زیر است:

```
Epoch 1/5
65/65 [==============================] - 49s 695ms/step - loss: 5.8263 - acc: 0.4096
Epoch 2/5
65/65 [==============================] - 43s 662ms/step - loss: 0.3991 - acc: 0.8867
Epoch 3/5
65/65 [==============================] - 44s 669ms/step - loss: 0.2798 - acc: 0.9177
Epoch 4/5
65/65 [==============================] - 43s 661ms/step - loss: 0.1639 - acc: 0.9642
Epoch 5/5
65/65 [==============================] - 43s 658ms/step - loss: 0.1462 - acc: 0.9698
<keras.callbacks.History at 0x7f0930d0a850>
```

نتایج روی داده‌های تست به شکل زیر است:

```
resnet.evaluate(test_generator)
```

```
33/33 [==============================] - 24s 678ms/step - loss: 5.0227 - acc: 0.4048
[5.022700309753418, 0.4048231542110443]
```

```
fine_tune_resnet.evaluate(test_generator)
```

```
33/33 [==============================] - 25s 710ms/step - loss: 0.1047 - acc: 0.9820
[0.1046600416302681, 0.981993556022644]
```

در فرایند آموزش در epoch های اول مدلی که با وزن‌های رندوم آموزش دیده‌است دقت بهتری نشان می‌دهد در حالی که در epoch اول دقت مدلی که وزن‌های imagenet را دارد پایین است. در epoch های بعدی دقت دو مدل تقریبا در یک حدود است.

روی داده‌های تست اما تفاوت زیادی را می‌بینیم، دقت مدل با وزن‌های imagenet روی این داده‌ها نسبت به مدل با وزن‌های رندوم تقریبا ۵۰ درصد بیشتر است. این نشان می‌دهد که گرچه مدل با وزن‌های رندوم به خوبی روی داده‌های train آموزش دیده‌است اما قابلیت تعمیم خوبی ندارد. مدل با وزن‌های imagenet اما قابلیت تعمیم خوبی نشان داده‌است. این را می‌توان به این دلیل دانست که وزن‌های pretrain مدل روی مجموعه وسیعی از داده‌ها آموزش دیده‌اند و به این دلیل مقادیر بهتر و مناسب‌تری پیدا کرده‌اند تا روی طیف وسیعی از داده‌ها جواب بدهند.

منابع:

https://jdhao.github.io/2017/11/06/resize-image-to-square-with-padding/
https://stackoverflow.com/questions/43120953/why-is-a-cnn-slower-to-train-than-a-fully-connected-mlp-in-keras
https://stackoverflow.com/questions/43120953/why-is-a-cnn-slower-to-train-than-a-fully-connected-mlp-in-keras
https://androidkt.com/explain-pooling-layers-max-pooling-average-pooling-global-average-pooling-and-global-max-pooling/
https://medium.com/geekculture/max-pooling-why-use-it-and-its-advantages-5807a0190459
https://alexisbcook.github.io/2017/global-average-pooling-layers-for-object-localization/
https://github.com/christianversloot/machine-learning-articles/blob/main/what-are-max-pooling-average-pooling-global-max-pooling-and-global-average-pooling.md
https://towardsdatascience.com/understanding-and-calculating-the-number-of-parameters-in-convolution-neural-networks-cnns-fc88790d530d#:~:text=Number%20of%20parameters%20in%20a%20CONV%20layer%20would%20be%20%3A%20((,1)*number%20of%20filters).