

به نام خدا



داده کاوی و یادگیری ماشین

گزارش تمرین چهارم

استاد درس :

جناب آقای دکتر زارع

دستیاران استاد :

سرکار خانم حسنی

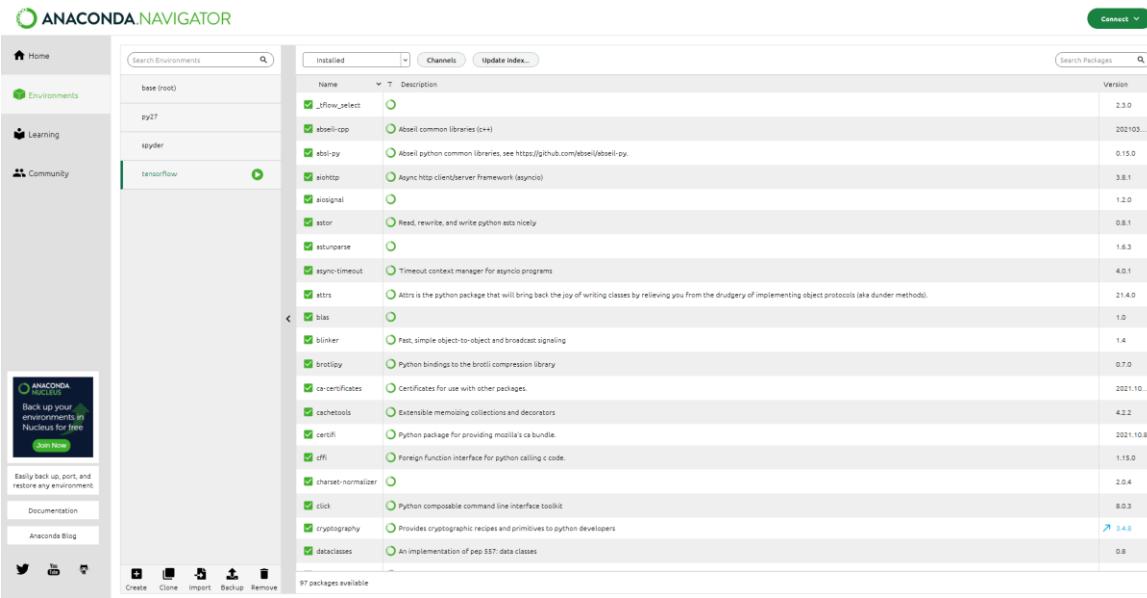
آقای داوردوست

آقای اسماعیلی

حسین رفیعی زاده

830400027

Setup TensorFlow and keras in Anaconda:

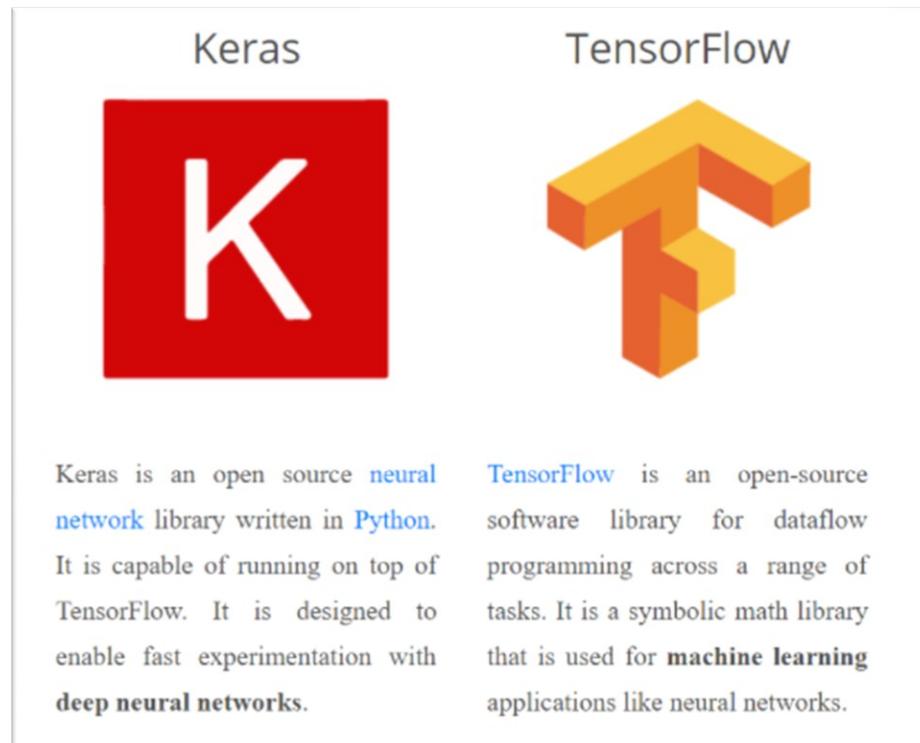


| Name | Description | Version |
|----------------------|---|---------|
| keras | Deep learning library for theano and tensorflow | 2.6.0 |
| opt_einsum | Optimizing einsum functions in numpy, tensorflow, dask, and more with contraction order optimization. | 3.3.0 |
| tensorboard | Tensorflow's visualization toolkit | 2.6.0 |
| tensorflow | Tensorflow is a machine learning library. | 2.6.0 |
| tensorflow-base | Tensorflow is a machine learning library, base package contains only tensorflow. | 2.6.0 |
| tensorflow-estimator | TensorFlow estimator is a high-level tensorflow api that greatly simplifies machine learning programming. | 2.6.0 |

```
Select Administrator: Anaconda Prompt (anaconda3) - conda install -c anaconda tensorflow-gpu
Found conflicts! Looking for incompatible packages.
This can take several minutes. Press CTRL-C to abort.
Examining sphinxcontrib-serializinghtml:  4%|███████████| 16/369 [00:45<23:42,  4.03s/it]
Examining bcrypt:  5%|███████████| 20/369 [00:46<10:56,  1.88s/it]
Examining pylint:  6%|███████████| 23/369 [00:56<08:31,  1.48s/it]
Examining mccabe:  8%|███████████| 30/369 [01:03<08:33,  1.51s/it]
Examining hdf5:  9%|███████████| 33/369 [01:03<07:49,  1.40s/it]
Examining babel: 11%|███████████| 42/369 [01:19<09:52,  1.81s/it]
Examining texdistance: 12%|███████████| 45/369 [01:20<05:45,  1.07s/it]
Examining freetype: 15%|███████████| 55/369 [01:21<01:34,  3.31it/s]
Examining flake8: 19%|███████████| 70/369 [01:36<05:53,  1.18s/it]
Examining binaryornot: 23%|███████████| 85/369 [01:46<02:21,  2.01it/s]
Examining backports.weakref: 24%|███████████| 87/369 [01:46<01:41,  2.79it/s]
Examining yaml: 25%|███████████| 93/369 [01:59<08:31,  1.85s/it]
Examining singledispatch: 35%|███████████| 129/369 [02:35<03:49,  1.04it/s]
Examining vc: 35%|███████████| 130/369 [02:36<03:11,  1.25it/s]
Examining daal4py: 37%|███████████| 136/369 [02:38<01:44,  2.24it/s]
Examining anaconda-project: 40%|███████████| 147/369 [02:51<04:10,  1.13s/it]
Examining imagesize: 41%|███████████| 151/369 [02:54<02:25,  1.50it/s]
Examining tblib: 41%|███████████| 153/369 [02:54<01:57,  1.83it/s]
Examining qtconsole: 41%|███████████| 153/369 [02:54<01:57,  1.83it/s]
Examining greenlet: 43%|███████████| 159/369 [02:57<01:39,  2.10it/s]
Examining watchdog: 44%|███████████| 164/369 [02:58<00:46,  4.37it/s]
Examining atomicwrites: 46%|███████████| 169/369 [03:02<01:56,  1.72it/s]
Examining json5: 46%|███████████| 171/369 [03:02<01:23,  2.37it/s]
Examining widgetsnbextension: 49%|███████████| 181/369 [03:09<02:17,  1.37it/s]
Examining tinyccss: 53%|███████████| 197/369 [03:26<02:48,  1.02it/s]
Examining @/win-64::_win==0=0: 54%|███████████| 201/369 [03:46<02:01,  1.39it/s]
Examining pep8: 60%|███████████| 223/369 [03:55<01:15,  1.93it/s]
Examining text-unidecode: 62%|███████████| 227/369 [03:57<01:20,  1.77it/s]
```

```
import numpy as np
import os
import seaborn as sn; sn.set(font_scale=1.4)
from sklearn.utils import shuffle
import matplotlib.pyplot as plt
import cv2
import tensorflow as tf
from tqdm import tqdm
import pandas as pd
from keras.callbacks import Callback
from sklearn.metrics import confusion_matrix, f1_score, precision_score, recall_score
```

کتابخانه های لازم را import میکنیم. Keras یک API یادگیری عمیق هستش که در پایتون نوشته شده است و دو کتابخانه یادگیری ماشین TensorFlow و Theano را پوشش میدهد و مهمترین مزیت این کتابخانه استفاده از مدل های آماده می باشد و سریع بودنش هست.



Keras vs TensorFlow

```
class_names = ['buildings', 'forest', 'glacier', 'mountain', 'sea', 'street']  
class_names_label = {class_name:i for i, class_name in enumerate(class_names)}
```

کلاس ها را مشخص میکنیم و طبق content زیر به هر کدام از کلاس ها یک label میدهیم.

```
{'buildings' → 0,  
'forest' → 1,  
'glacier' → 2,  
'mountain' → 3,  
'sea' → 4,  
'street' → 5 }
```

content

```
DESKTOP_J  
{'buildings': 0, 'forest': 1, 'glacier': 2, 'mountain': 3, 'sea': 4, 'street': 5}
```

خروجی

```

def load_data():

    datasets = ['input/seg_train/seg_train', 'input/seg_test/seg_test']
    output = []

    # Iterate through training and test sets
    for dataset in datasets:

        images = []
        labels = []

        print("Loading {}".format(dataset))

        # Iterate through each folder corresponding to a category
        for folder in os.listdir(dataset):
            label = class_names_label[folder]

            # Iterate through each image in our folder
            for file in tqdm(os.listdir(os.path.join(dataset, folder))):

                # Get the path name of the image
                img_path = os.path.join(os.path.join(dataset, folder), file)

                # Open and resize the img
                image = cv2.imread(img_path)
                image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
                image = cv2.resize(image, IMAGE_SIZE)

                # Append the image and its corresponding label to the output
                images.append(image)
                labels.append(label)

        images = np.array(images, dtype = 'float32')
        labels = np.array(labels, dtype = 'int32')

        output.append((images, labels))

    return output

```

ابتدا باید دیتاست را Load کنیم برای این کار تابعی به نام Load_data تعریف میکنیم که تصاویر و label ها را از فایل بارگذاری کند.

```
(train_images, train_labels), (test_images, test_labels) = load_data()
```

حالا تابع Load_data را فراخوانی میکنیم و داده ها را به چهار بخش train_images, train_labels تقسیم میکنیم.

```
Loading input/seg_train/seg_train
100%|██████████| 2191/2191 [00:02<00:00, 864.85it/s]
100%|██████████| 2271/2271 [00:03<00:00, 720.65it/s]
100%|██████████| 2404/2404 [00:02<00:00, 838.99it/s]
100%|██████████| 2512/2512 [00:03<00:00, 745.80it/s]
100%|██████████| 2274/2274 [00:02<00:00, 914.27it/s]
100%|██████████| 2382/2382 [00:02<00:00, 879.38it/s]
Loading input/seg_test/seg_test
100%|██████████| 437/437 [00:00<00:00, 567.54it/s]
100%|██████████| 474/474 [00:00<00:00, 591.14it/s]
100%|██████████| 553/553 [00:00<00:00, 878.91it/s]
100%|██████████| 525/525 [00:00<00:00, 905.31it/s]
100%|██████████| 510/510 [00:00<00:00, 873.41it/s]
100%|██████████| 501/501 [00:00<00:00, 815.65it/s]
```

Loading inputs

```

n_train = train_labels.shape[0]
n_test = test_labels.shape[0]

print ("Number of training examples: {}".format(n_train))
print ("Number of testing examples: {}".format(n_test))
print ("Each image is of size: {}".format(IMAGE_SIZE))

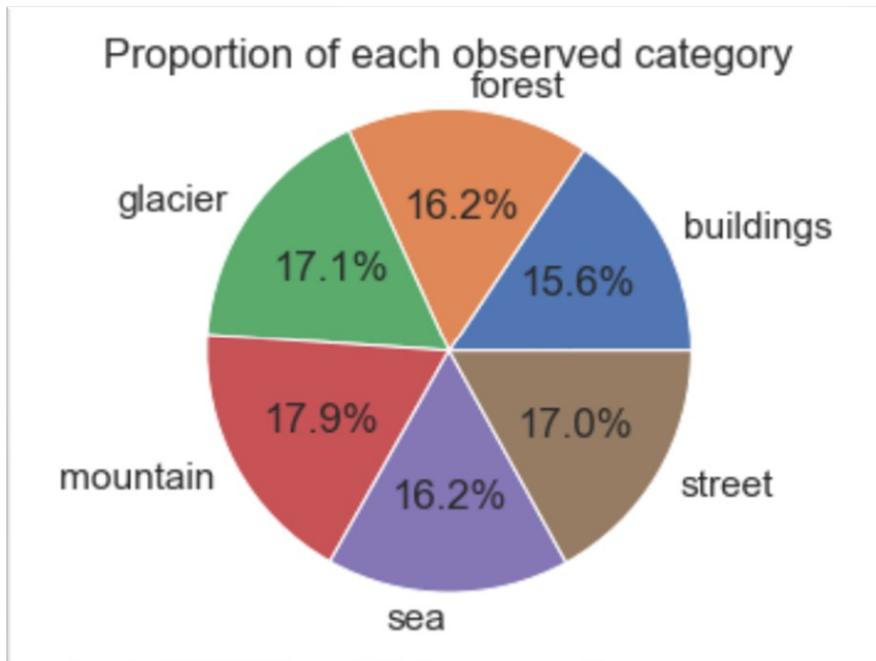
```

```

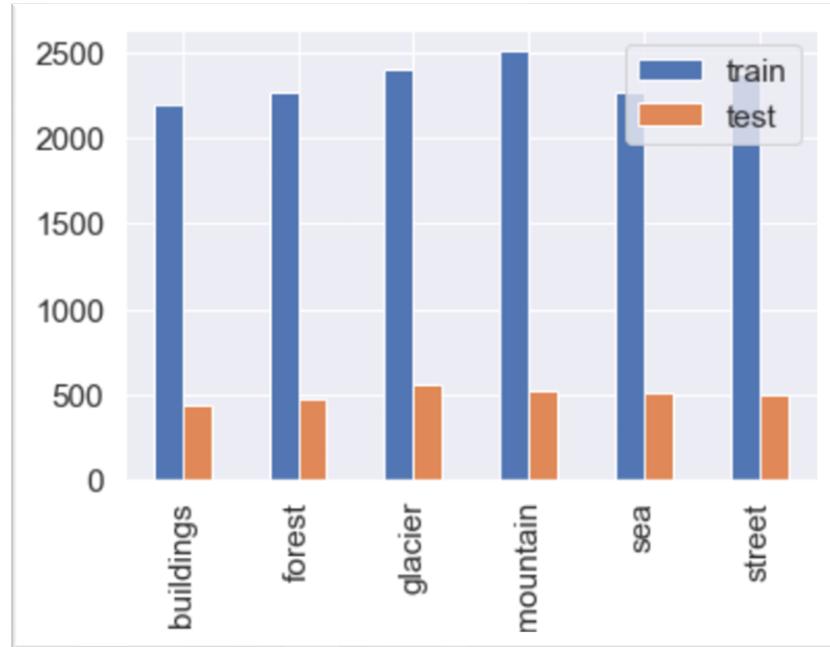
Number of training examples: 14034
Number of testing examples: 3000
Each image is of size: (150, 150)

```

14034 داده آموزشی و 3000 داده تست داریم و اندازه هر تصویر برابر است با 150 در 150 پیکسل .



میزان دیتا ها را در هر کلاس مشاهده میکنیم.



میزان داده های train,test را در هر کلاس مشاهده میکنیم.

```
[[[ [198. 213. 234.]
    [190. 206. 229.]
    [180. 196. 221.]
    ...
    [133. 164. 208.]
    [137. 166. 210.]
    [137. 166. 208.]]

   [[190. 205. 228.]
    [183. 199. 224.]
    [173. 190. 216.]
    ...
    [132. 163. 209.]
    [135. 166. 210.]]]
```

تصویر یک آرایه یا ماتریس دو بعدی از اعداد هست که هر کدام از درایه های آن معادل یک پیکسل هست، مقادیر موجود در تصویر یا ماتریس بین ۰ تا ۲۵۵ هستند و هر چه این مقادیر به صفر نزدیک تر باشند، آن پیکسل تیره تر میشود و هر چه قدر مقادیر به ۲۵۵ نزدیک تر باشد رنگ آن پیکسل روشن تر میشود.

```
train_images = train_images / 255.0  
test_images = test_images / 255.0
```

تمامی این مقادیر باید نرمال سازی شوند و بین 0 تا 1 قرار بگیرند تا محاسبات ساده‌تر گردد.

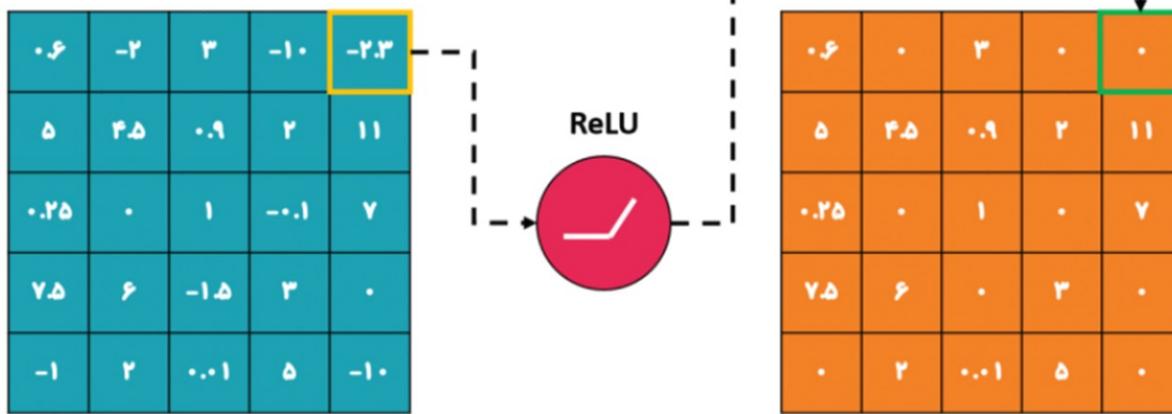
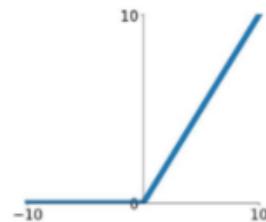
```
Each image is of size: (150, 150,  
[[[0.7764706 0.8352941 0.91764706]  
[0.74509805 0.80784315 0.8980392 ]  
[0.7058824 0.76862746 0.8666667 ]  
...  
[0.52156866 0.6431373 0.8156863 ]  
[0.5372549 0.6509804 0.8235294 ]  
[0.5372549 0.6509804 0.8156863 ]]  
  
[[0.74509805 0.8039216 0.89411765]  
[0.7176471 0.78039217 0.8784314 ]  
[0.6784314 0.74509805 0.84705883]  
...  
[0.5176471 0.6392157 0.81960785]
```

خروجی

```
tf.keras.layers.Conv2D(32, (3, 3), activation = 'relu', padding='same', input_shape = (150, 150, 3)),
```

جواب سوال 3 قسمت a) برای طراحی شبکه عصبی کانولوشن از 6 لایه استفاده کردیم که دو لایه آن کانولوشنی می باشد ، معمولا در شبکه عصبی کانولوشن، ابتدا لایه کانولوشنی با تعداد فیلتر کم (32) قرار میگیرد. معمولا بعد از لایه کانولوشنی یک Activation Function قرار میگیرد چون که در نورون هم بعد از جمع وزن دهی و جمع با بایاس، Activation Function اعمال می شد. تابع ReLU(Rectified Linear Unit) روی Feature Map (خروجی لایه کانولوشنی) اعمال می شود. کارش هم اینست که منفی ها را صفر می کند، در بین تمام توابع غیرخطی، تابع ReLU بیشترین محبوبیت را دارد و برای تصاویر اکثرا از این تابع استفاده میکنیم چون که مزیت اصلی این تابع این هست که همه نورون ها را به صورت همزمان فعال نمیکند. در ضمن اندازه ورودی ما $150 \times 150 \times 3$ و عدد 3 هم بخاطر RGB میباشد.

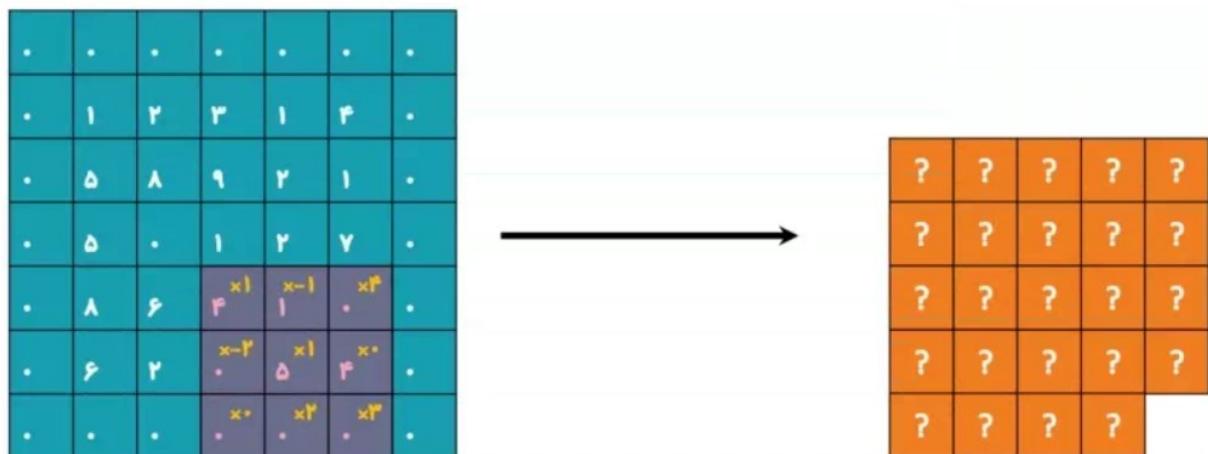
ReLU
 $\max(0, x)$



تابع تحریک ReLU و نحوه اعمال به یک ورودی نمونه

جواب سوال 3 قسمت b

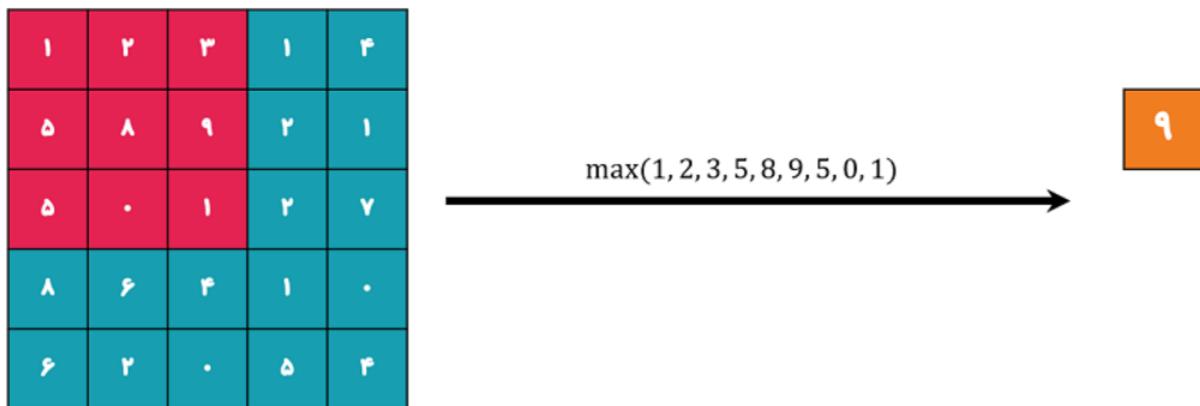
Padding: به عنوان مثال بعد از فیلتر کردن یک ماتریس 5×3 با فیلتر 3×5 با سایز خروجی دو سطر و ستون کمتر از ماتریس ورودی می شود، با لایه گذاری یا padding می توانیم به صورت جعلی اندازه ورودی را افزایش دهیم تا ماتریس خروجی همان اندازه ماتریس ورودی بشود. یک راه ساده و رایج آن اضافه کردن سطر و ستون صفر به صورت متقابله به دور ماتریس ورودی است. به لایه گذاری صفر zero padding گفته می شود. در اینجا به مقدار پدینگ same را داده ایم تا سایز ورودی با خروجی یکی شود.



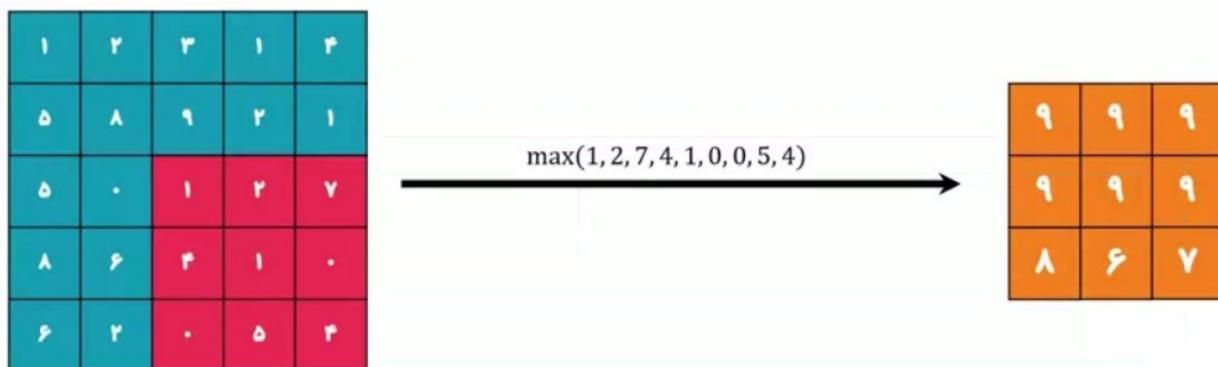
کانولوشن با padding

```
tf.keras.layers.MaxPooling2D(2,2),
```

لایه بعدی Pooling هستش، لایه پولینگ یکی از مهمترین لایه های شبکه عصبی کانولوشن می باشد. هدف لایه pooling کاهش اندازه مکانی Feature Map هست و لایه پولینگ پارامتر قابل آموزش ندارد صرفا یک نمونه برداری ساده و موثر انجام میدهد و عملکردش شبیه کانولوشن هست و یک پنجره روی تصویر حرکت میکند. رایج ترین نمونه پولینگ average pooling و max pooling است که در اینجا از max استفاده شده است.



نحوه عملکرد max pooling در شبکه عصبی کانولوشن

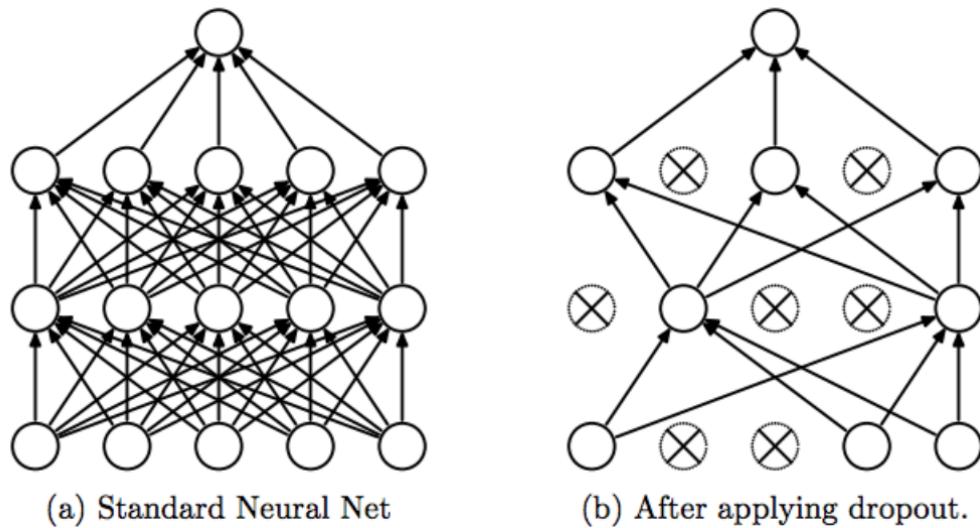


عملکرد max pooling در شبکه عصبی کانولوشن

جواب سوال 3 قسمت (c)

```
tf.keras.layers.Dropout(0.25),
```

در Deep Learning برای جلوگیری از overfitting Data از تکنیک های augmentation استفاده می شود. معمولا برای کاهش overfitting مدل در شبکه های CNN از عملگر Dropout استفاده می شود. Dropout باید در هنگام train بر روی اتصالات بین لایه ها اعمال شوند و در هنگام تست و ارزیابی شبکه نباید اعمال گردد. در ضمن اضافه کردن فیلتر تا حدی خوب می باشد و اگر بیش از حد تعدادشان و سایزشان بالا رود، باعث overfitting می شود.



عملگر DropOut بین لایه های fullyconnected قرار می گیرد. اینکه عملگر DropOut را بین کدام یک از این لایه ها قرار دهیم و تعیین احتمال صحیح DropOut معمولا به صورت تجربی است. اما قاعدهاً بهتر است بین لایه هایی که بیشترین اتصالات در آن ها وجود دارد اعمال شود.

```
tf.keras.layers.Conv2D(64, (3, 3), activation = 'relu',padding='same'),  
tf.keras.layers.MaxPooling2D(2,2),  
tf.keras.layers.Dropout(0.25),
```

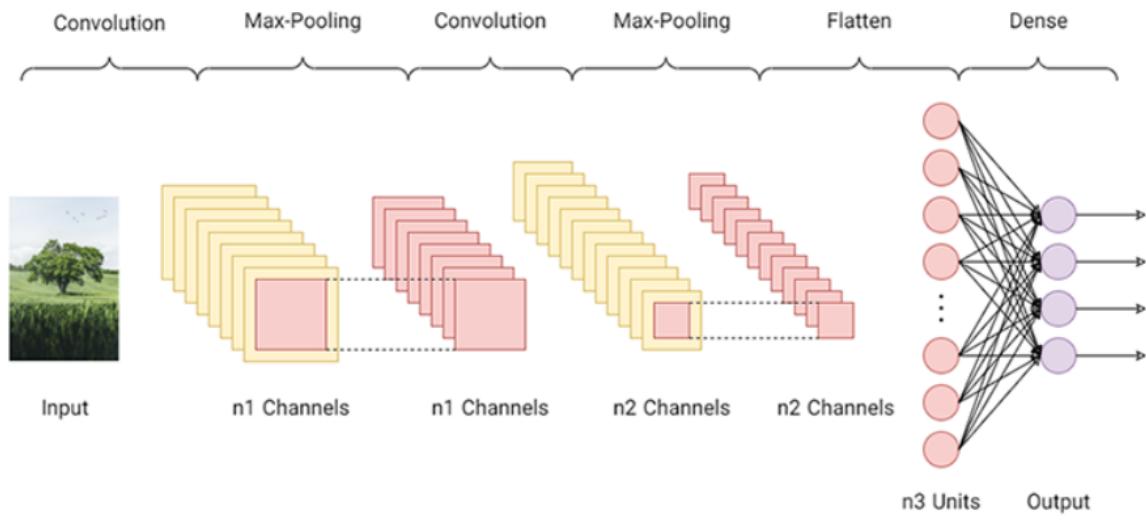
برای سه لایه بعد هم به همین صورت می باشد فقط به جای 32 فیلتر 64 فیلتر قرار میدهیم چون ما جزئیات بیشتر و متنوع تری را میخواهیم تولید کنیم با خاطر همین رفتہ تعداد فیلترها را بیشتر میکنیم.

```
tf.keras.layers.Flatten(),
```

در اینجا همه ی لایه ها را با هم ادغام میکنیم و تبدیل به یک لایه میکنیم.

```
tf.keras.layers.Dense(128, activation=tf.nn.relu),  
tf.keras.layers.Dense(6, activation=tf.nn.softmax)
```

خروجی می تواند یک لایه softmax باشد که نشان دهد تصویر مربوط به یک دریا است یا چیز دیگر، همچنین می توانید یک لایه sigmoid داشته باشید تا احتمال دریا بودن تصویر را به شما بدهد. که ما در اینجا خروجی softmax را در نظر گرفتیم.



CNN Model

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

Optimizer: adam = RMSProp + Momentum.

: برای داشتن به روز رسانی های بهتر گرادیان را در نظر میگرید

: میانگین وزنی مربع های گرادیان گذشته RMSProp

: Loss function

از sparse categorical crossentropy برای دسته بندی استفاده میکنیم و هر تصویر فقط به یک کلاس متعلق است.

```
history = model.fit(train_images, train_labels, epochs=20, validation_split = 0.2)
```

مدل را با داده های آموزشی فیت میکنیم تا شبکه عصبی الگو ها را learn کنه تا بعدا بتونه کلاس هر کدام از عکس هارا تشخیص دهد.

```
Epoch 1/20
351/351 [=====] - 37s 73ms/step - loss: 1.4628 - accuracy: 0.5110 - val_loss: 1.0139 - val_accuracy: 0.6213
Epoch 2/20
351/351 [=====] - 25s 71ms/step - loss: 0.8493 - accuracy: 0.6837 - val_loss: 0.7692 - val_accuracy: 0.7235
Epoch 3/20
351/351 [=====] - 24s 69ms/step - loss: 0.6246 - accuracy: 0.7724 - val_loss: 0.7044 - val_accuracy: 0.7503
Epoch 4/20
351/351 [=====] - 24s 69ms/step - loss: 0.4530 - accuracy: 0.8396 - val_loss: 0.6310 - val_accuracy: 0.7823
Epoch 5/20
351/351 [=====] - 24s 69ms/step - loss: 0.3284 - accuracy: 0.8872 - val_loss: 0.6506 - val_accuracy: 0.7823
Epoch 6/20
351/351 [=====] - 25s 72ms/step - loss: 0.2290 - accuracy: 0.9217 - val_loss: 0.7012 - val_accuracy: 0.7777
Epoch 7/20
351/351 [=====] - 24s 69ms/step - loss: 0.1599 - accuracy: 0.9455 - val_loss: 0.8224 - val_accuracy: 0.7813
Epoch 8/20
351/351 [=====] - 24s 69ms/step - loss: 0.1118 - accuracy: 0.9623 - val_loss: 0.8253 - val_accuracy: 0.7976
Epoch 9/20
351/351 [=====] - 24s 69ms/step - loss: 0.0904 - accuracy: 0.9711 - val_loss: 0.8784 - val_accuracy: 0.7944
Epoch 10/20
351/351 [=====] - 25s 71ms/step - loss: 0.0724 - accuracy: 0.9765 - val_loss: 0.8749 - val_accuracy: 0.8055
Epoch 11/20
351/351 [=====] - 24s 69ms/step - loss: 0.0633 - accuracy: 0.9794 - val_loss: 0.9196 - val_accuracy: 0.7919
Epoch 12/20
351/351 [=====] - 24s 69ms/step - loss: 0.0443 - accuracy: 0.9864 - val_loss: 1.0151 - val_accuracy: 0.7820
Epoch 13/20
351/351 [=====] - 24s 69ms/step - loss: 0.0456 - accuracy: 0.9857 - val_loss: 1.1100 - val_accuracy: 0.7627
Epoch 14/20
351/351 [=====] - 24s 69ms/step - loss: 0.0516 - accuracy: 0.9828 - val_loss: 1.2495 - val_accuracy: 0.7706
Epoch 15/20
351/351 [=====] - 24s 68ms/step - loss: 0.0372 - accuracy: 0.9882 - val_loss: 1.0856 - val_accuracy: 0.7823
Epoch 16/20
351/351 [=====] - 24s 69ms/step - loss: 0.0466 - accuracy: 0.9850 - val_loss: 1.1668 - val_accuracy: 0.7748
Epoch 17/20
351/351 [=====] - 24s 69ms/step - loss: 0.0510 - accuracy: 0.9837 - val_loss: 1.1688 - val_accuracy: 0.7937
Epoch 18/20
351/351 [=====] - 24s 68ms/step - loss: 0.0428 - accuracy: 0.9866 - val_loss: 1.1882 - val_accuracy: 0.7802
- 1/100

Epoch 19/20
351/351 [=====] - 24s 69ms/step - loss: 0.0260 - accuracy: 0.9921 - val_loss: 1.2067 - val_accuracy: 0.7802
Epoch 20/20
351/351 [=====] - 24s 69ms/step - loss: 0.0402 - accuracy: 0.9887 - val_loss: 1.2374 - val_accuracy: 0.7909
```

هر مرحله که جلوتر میرویم مدل آموزش بیشتری میبینه و مقدار accuracy بهتر میشه.

```
test_loss = model.evaluate(test_images, test_labels)
```

حالا باید عملکرد مدل را در مجموعه تست ارزیابی کنیم.

```
94/94 [=====] - 2s 23ms/step - loss: 1.3152 - accuracy: 0.7817
```

مقدار accuracy روی داده تست برابر با 0.78 شد.

```
def plot_accuracy_loss(history):
    """
        Plot the accuracy and the Loss during the training of the nn.
    """
    fig = plt.figure(figsize=(10,5))

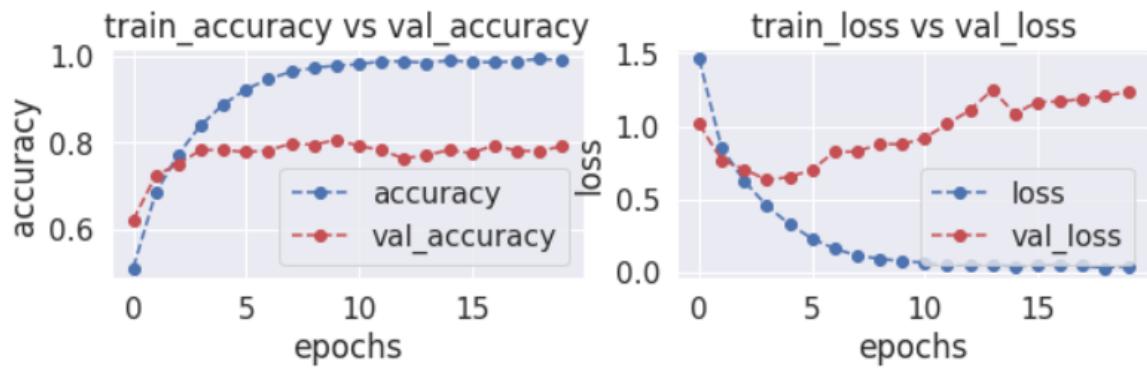
    # Plot accuracy
    plt.subplot(221)
    plt.plot(history.history['accuracy'], 'bo--', label = "accuracy")
    plt.plot(history.history['val_accuracy'], 'ro--', label = "val_accuracy")
    plt.title("train_accuracy vs val_accuracy")
    plt.ylabel("accuracy")
    plt.xlabel("epochs")
    plt.legend()

    # Plot loss function
    plt.subplot(222)
    plt.plot(history.history['loss'], 'bo--', label = "loss")
    plt.plot(history.history['val_loss'], 'ro--', label = "val_loss")
    plt.title("train_loss vs val_loss")
    plt.ylabel("loss")
    plt.xlabel("epochs")

    plt.legend()
    plt.show()

plot_accuracy_loss(history)
```

رسم نمودار accuracy



هر epoch که جلوتر میرویم مدل بهتر میشود و نزدیک به یک میشود ولی هیچگاه خود یک نمیشود. (train)

```
from sklearn.metrics import classification_report
print(classification_report(test_labels, pred_labels, target_names=class_names))
```

F1-score, recall and precision محاسبه

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| buildings | 0.77 | 0.68 | 0.72 | 437 |
| forest | 0.93 | 0.95 | 0.94 | 474 |
| glacier | 0.71 | 0.77 | 0.74 | 553 |
| mountain | 0.73 | 0.73 | 0.73 | 525 |
| sea | 0.80 | 0.73 | 0.76 | 510 |
| street | 0.78 | 0.83 | 0.80 | 501 |
| accuracy | | | 0.78 | 3000 |
| macro avg | 0.79 | 0.78 | 0.78 | 3000 |
| weighted avg | 0.78 | 0.78 | 0.78 | 3000 |

F1 score, recall and precision in without Augmentation

```
def display_random_image(class_names, images, labels):
    """
    Display a random image from the images array and its correspond label from the labels array.
    """
    index = np.random.randint(images.shape[0])
    plt.figure()
    plt.imshow(images[index])
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.title('Image #{} : {}'.format(index, class_names[labels[index]]))
    plt.show()
```

در این قسمت ما میتوانیم یک تصویر تصادفی را از مجموعه آموزشی نمایش دهیم.

```
predictions = model.predict(test_images)      # Vector of probabilities
pred_labels = np.argmax(predictions, axis = 1) # We take the highest probability
display_random_image(class_names, test_images, pred_labels)
```

در اینجا ما میخواهیم کلاس تصویر را پیش بینی کنیم و ببینم که تصویر در کدام یک از 6 دسته قرار میگیرد. predictions مجموعه ای از احتمالات را به ما میدهد که مشخص میکند هر تصویر با چه احتمالی به هر کدام از کلاس ها منتبه هست سپس ما بیشترین احتمال را پیدا میکنیم سپس نام کلاسی که بیشترین احتمال را دارد به همراه اسم کلاس ها و test_image را به تابع display_random_image میدهیم تا تصویر را برای ما پیدا و چاپ نماید.

Image #2860 : sea



Image #966 : glacier



Image #1744 : mountain

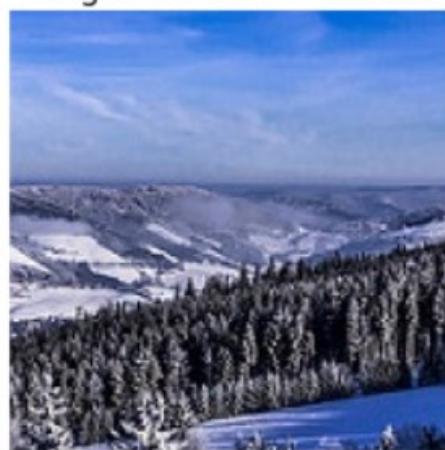
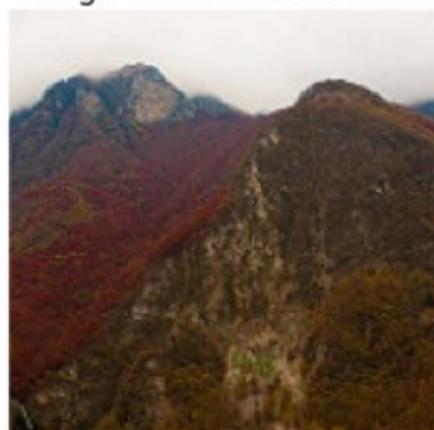


Image #1513 : mountain



خروجي

```

def display_examples(class_names, images, labels):
    """
        Display 25 images from the images array with its corresponding labels
    """

    fig = plt.figure(figsize=(10,10))
    fig.suptitle("Some examples of images of the dataset", fontsize=16)
    for i in range(25):
        plt.subplot(5,5,i+1)
        plt.xticks([])
        plt.yticks([])
        plt.grid(False)
        plt.imshow(images[i], cmap=plt.cm.binary)
        plt.xlabel(class_names[labels[i]])
    plt.show()

```

تابع display_examples از 25 تصویر در images array را به همراه لیبل هاشون نمایش میدهد.

Error analysis:

```

def print_mislabeled_images(class_names, test_images, test_labels, pred_labels):
    """
        Print 25 examples of mislabeled images by the classifier, e.g when test_labels != pred_labels
    """
    B00 = (test_labels == pred_labels)
    mislabeled_indices = np.where(B00 == 0)
    mislabeled_images = test_images[mislabeled_indices]
    mislabeled_labels = pred_labels[mislabeled_indices]

    title = "Some examples of mislabeled images by the classifier:"
    display_examples(class_names, mislabeled_images, mislabeled_labels)

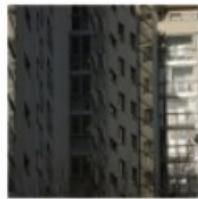
print_mislabeled_images(class_names, test_images, test_labels, pred_labels)

```

از طریق این تابع ما میتوانیم بفهمیم که کلاس بندی در کدام تصاویر(فقط در 25 تصویر) مشکل دارد و کلاس آن ها به درستی predict نمیشود و لیبل آن ها اشتباه می باشد.



street



forest



sea



sea



street



forest



sea



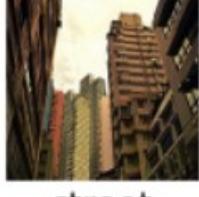
glacier



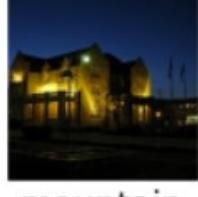
street



street



street



mountain



sea



glacier



street



street



street



street



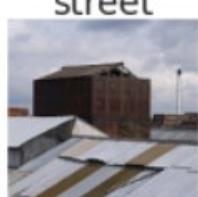
mountain



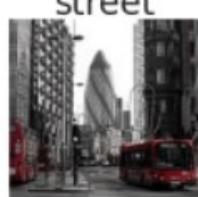
street



street



sea



street



street



street

خروجي

Augmentation:

```
datagen=ImageDataGenerator(  
    rotation_range=15,  
    horizontal_flip=True,  
    width_shift_range=0.1,  
    height_shift_range=0.1  
  
)  
  
datagen.fit(train_images)
```

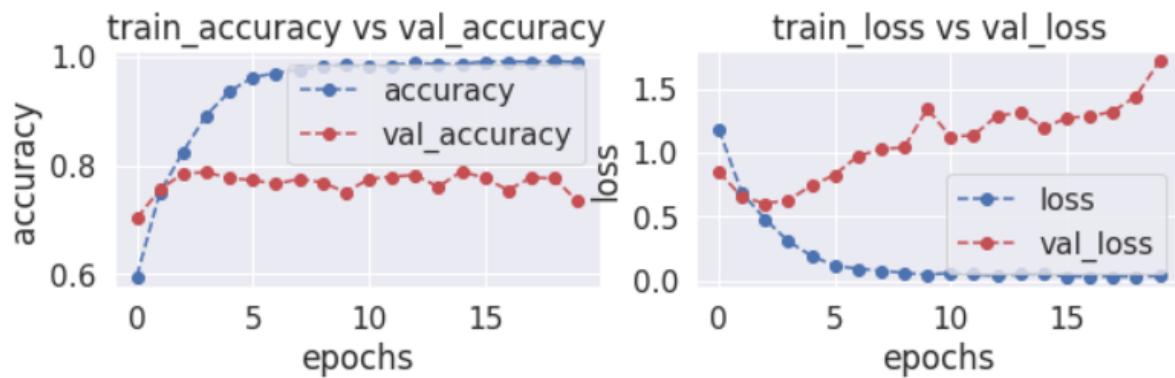
تعداد داده ورودی را افزایش میدهد به طور مثال از طریق شیفت دادن پیکسل ها. ابتدا ما او مدیم محدوده چرخش را مشخص کردیم و برابر با 15 گذاشتیم سپس horizontal_flip را برابر true گذاشتیم که به طور تصادفی میاد ورودی ها را عمودی میکند سپس محدوده شیفت افقی و عمودی را مشخص کردیم و برابر با 0.1 قرار دادیم سپس بر روی train_images فیت کردیم و بقیه مراحل مانند without Augmentation می باشد.

```
Epoch 1/20  
351/351 [=====] - 36s 71ms/step - loss: 1.1860 - accuracy: 0.5954 - val_loss: 0.8510 - val_accuracy: 0.7025  
Epoch 2/20  
351/351 [=====] - 23s 66ms/step - loss: 0.6870 - accuracy: 0.7503 - val_loss: 0.6542 - val_accuracy: 0.7556  
Epoch 3/20  
351/351 [=====] - 23s 67ms/step - loss: 0.4815 - accuracy: 0.8250 - val_loss: 0.6022 - val_accuracy: 0.7848  
Epoch 4/20  
351/351 [=====] - 24s 67ms/step - loss: 0.3079 - accuracy: 0.8920 - val_loss: 0.6299 - val_accuracy: 0.7873  
Epoch 5/20  
351/351 [=====] - 23s 67ms/step - loss: 0.1946 - accuracy: 0.9357 - val_loss: 0.7410 - val_accuracy: 0.7763  
Epoch 6/20  
351/351 [=====] - 23s 66ms/step - loss: 0.1176 - accuracy: 0.9614 - val_loss: 0.8261 - val_accuracy: 0.7727  
Epoch 7/20  
351/351 [=====] - 23s 66ms/step - loss: 0.0935 - accuracy: 0.9694 - val_loss: 0.9717 - val_accuracy: 0.7656  
Epoch 8/20  
351/351 [=====] - 23s 66ms/step - loss: 0.0774 - accuracy: 0.9754 - val_loss: 1.0263 - val_accuracy: 0.7748  
Epoch 9/20  
351/351 [=====] - 24s 69ms/step - loss: 0.0609 - accuracy: 0.9815 - val_loss: 1.0402 - val_accuracy: 0.7684  
Epoch 10/20  
351/351 [=====] - 23s 66ms/step - loss: 0.0473 - accuracy: 0.9841 - val_loss: 1.3420 - val_accuracy: 0.7506  
Epoch 11/20  
351/351 [=====] - 23s 66ms/step - loss: 0.0610 - accuracy: 0.9804 - val_loss: 1.1217 - val_accuracy: 0.7745  
Epoch 12/20  
351/351 [=====] - 23s 66ms/step - loss: 0.0512 - accuracy: 0.9834 - val_loss: 1.1364 - val_accuracy: 0.7791  
Epoch 13/20  
351/351 [=====] - 23s 66ms/step - loss: 0.0377 - accuracy: 0.9884 - val_loss: 1.2862 - val_accuracy: 0.7820  
Epoch 14/20  
351/351 [=====] - 23s 66ms/step - loss: 0.0518 - accuracy: 0.9851 - val_loss: 1.3112 - val_accuracy: 0.7602  
Epoch 15/20  
351/351 [=====] - 23s 66ms/step - loss: 0.0516 - accuracy: 0.9858 - val_loss: 1.1931 - val_accuracy: 0.7891  
Epoch 16/20  
351/351 [=====] - 23s 66ms/step - loss: 0.0327 - accuracy: 0.9905 - val_loss: 1.2696 - val_accuracy: 0.7763  
Epoch 17/20  
351/351 [=====] - 23s 66ms/step - loss: 0.0332 - accuracy: 0.9903 - val_loss: 1.2826 - val_accuracy: 0.7545  
Epoch 18/20  
351/351 [=====] - 23s 66ms/step - loss: 0.0324 - accuracy: 0.9901 - val_loss: 1.3205 - val_accuracy: 0.7773  
Epoch 19/20  
351/351 [=====] - 23s 67ms/step - loss: 0.0319 - accuracy: 0.9911 - val_loss: 1.4401 - val_accuracy: 0.7759  
Epoch 20/20  
351/351 [=====] - 23s 66ms/step - loss: 0.0371 - accuracy: 0.9894 - val_loss: 1.7125 - val_accuracy: 0.7335
```

هر مرحله که جلوتر میرویم مدل اموزش بیشتری میبینه و مقدار accuracy بهتر میشه.

```
94/94 [=====] - 2s 23ms/step - loss: 1.7040 - accuracy: 0.7357
```

مقدار accuracy روی داده تست برابر با 0.73 شد.



Accuracy and loss function with Augmentation
(train)

```
from sklearn.metrics import classification_report
print(classification_report(test_labels, pred_labels, target_names=class_names))
```

محاسبه F1-score, recall and precision

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| buildings | 0.67 | 0.79 | 0.72 | 437 |
| forest | 0.98 | 0.82 | 0.89 | 474 |
| glacier | 0.72 | 0.62 | 0.67 | 553 |
| mountain | 0.79 | 0.58 | 0.67 | 525 |
| sea | 0.61 | 0.82 | 0.70 | 510 |
| street | 0.76 | 0.80 | 0.78 | 501 |
| accuracy | | | 0.74 | 3000 |
| macro avg | 0.75 | 0.74 | 0.74 | 3000 |
| weighted avg | 0.75 | 0.74 | 0.74 | 3000 |

F1-score,recall and precision with Augmentation