Deep Learning
Dr. DavoodAbadi
Fall 2022
Hoorieh Sabzevari - 98412004
HW4

.۱

محاسبات:

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 10 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix} = -80+10 = -70$$

$$\begin{bmatrix} 0 & 10 & 0 \\ 0 & 10 & 0 \\ 0 & 10 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix} = -80+20 = -60$$

$$\begin{bmatrix} 10 & 0 & 0 \\ 10 & 0 & 0 \\ 10 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix} = 30$$

$$\begin{bmatrix} 0 & 0 & 0 \\ 10 & 0 & 0 \\ 10 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix} = 20$$

خروجی:

0	20	-70	20	0
0	30	-60	30	0
0	30	-60	30	0
0	30	-60	30	0
0	20	-70	20	0

ماتریس اولیه در تمامی نقاط بجز ستون وسط صفر است. این ماتریس حکم یک خط را دارد. اما با اعمال کانولوشن، سه ستون از ماتریس اولیه مقدار غیرصفر می‌گیرند. در واقع نسبت به تغییرات شدید حساس است.

۲. یک کانولوشن 1×1 را شبکه در شبکه نیز می نامند و در بسیاری از مدل‌های CNN مانند مدل‌های ResNet و Inception بسیار مفید است. کاری که این کانولوشن انجام می‌دهد صرفا ضرب یک عدد در مقادیر ماتریس اولیه نیست. برای درک بهتر ابتدا یک مثال را در نظر بگیریم:

ورودی: $1 \times 6 \times 6$ ، فیلتر: یک فیلتر $1 \times 1 \times 1$ ، خروجی: $1 \times 6 \times 6$

مثالی دیگر:

ورودی: $32 \times 6 \times 6$ ، فیلتر: ۵ فیلتر $32 \times 1 \times 1$ ، خروجی: $5 \times 6 \times 6$

کانولوشن ۱×۱ زمانی مفید است که:

ما می خواهیم تعداد کانال‌ها را کاهش دهیم. به این ویژگی تبدیل feature transformation نیز می‌گویند. در مثال دوم بالا، ورودی را از ۳۲ به ۵ کانال کاهش دادیم. همچنین با کوچک کردن عمق می‌توانیم محاسبات زیادی را ذخیره کنیم.

اگر تعداد فیلترهای 1×1 را با تعداد کانال های ورودی یکسان تعیین کرده باشیم، خروجی شامل همان تعداد کانال خواهد بود. سپس $conv\ 1 \times 1$ مانند یک غیر خطی عمل می‌کند و عملگر غیرخطی را یاد می‌گیرد.

لایه های کاملاً متصل را با کانولوشن های ۱×۱ جایگزین کرد زیرا Yann LeCun معتقد است که آنها یکسان هستند. در Convolutional Nets چیزی به نام "لایه های کاملا متصل" وجود ندارد. فقط لایه های کانولوشن با هسته های کانولوشن 1×1 و یک جدول اتصال کامل وجود دارد و به همین اصطلاح شبکه در شبکه برای آن به کار برده می‌شود. زیرا فیلتر 1×1 با عمق دلخواه در همان بردار تصویر ضرب شده

۳.

الف) padding = same یعنی zero-padding داریم و ابعاد ورودی تغییر پیدا نمی‌کند.

پس طول و عرض خروجی فقط توسط stride نصف می‌شود و برابر با $14 \times 14 \times 32$ می‌شود.

ب) $7 \times 7 \times 32$ ، زیرا max-pooling و stride همزمان باعث نصف شدن سطر و ستون می‌شوند.

ج) ابعاد خروجی لایه‌ی Flatten برابر با ۱۵۶۸ نورون است که حاصل‌ضرب ابعاد خروجی لایه‌ی قبل است. این لایه پارامتری ندارد.

ابعاد لایه‌ی خروجی نیز برابر با ۵ نورون است. زیرا مسئله‌ی ما دسته‌بندی ۵کلاسه است. تعداد پارامترهای آن هم به صورت زیر است:

$$5 \times (1568 + 1) = 7845$$

.۴

Forward pass:

$$X * F = O$$

$$O_{11} = 2 \times 0 + 3 \times 3 + 3 \times 1 + 1 \times (-2) = 10$$

$$O_{21} = 3 \times 0 + 1 \times 3 + 4 \times 1 + (-1) \times (-2) = 9$$

$$O_{12} = 3 \times 0 + 4 \times 3 + 1 \times 1 + 5 \times (-2) = 3$$

$$O_{22} = 1 \times 0 + 5 \times 3 + (-1) \times 1 + (-2) \times (-2) = 18$$

$$O = \begin{bmatrix} 10 & 3 \\ 9 & 18 \end{bmatrix}$$

$$O = GAP = \frac{10 + 3 + 9 + 18}{4} = 10$$

Backward pass:

$$\frac{\partial L}{\partial O} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \qquad \frac{\partial L}{\partial F} = \frac{\partial L}{\partial O} * \frac{\partial O}{\partial F}, \qquad \frac{\partial O}{\partial F} = X$$

$$= \frac{\partial L}{\partial O} * X = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} * \begin{bmatrix} 2 & 3 & 4 \\ 3 & 1 & 5 \\ 4 & -1 & -2 \end{bmatrix} = \begin{bmatrix} 9 & 13 \\ 7 & 3 \end{bmatrix}$$

۴. ابتدا عکس‌ها را لود کرده و سپس دایرکتوری‌های مربوط به سه مجموعه‌ی train و test و validation را می‌سازیم. سپس با نسبتی دلخواه عکس‌ها را داخل دایرکتوری‌ها کپی می‌کنیم. در اینجا ۸۰٪ را برای داده‌های آموزشی، ۱۰٪ برای داده‌های تست و ۱۰٪ هم برای داده‌های اعتبارسنجی جدا می‌کنیم.

```
4]  path='/content/dataset'
    classes = os.listdir(path)
```

```
    os.mkdir("data")

    os.mkdir("data/train")
    os.mkdir("data/test")
    os.mkdir("data/validation")

    for name in classes:
      os.mkdir(f"data/train/{name}")
      os.mkdir(f"data/validation/{name}")
      os.mkdir(f"data/test/{name}")
```

```
6]  for class_name in classes:
        images = os.listdir(path + f"/{class_name}")
        l = len(images)
        train_images = random.sample(images, int(0.8 * l))
        temp = list(set(images).difference(set(train_images)))
        l = len(temp)
        val_images = random.sample(temp, int(0.5 * l))
        test_images = list(set(temp).difference(set(val_images)))

        for image in train_images:
          shutil.copy(path + f"/{class_name}/{image}", f"data/train/{class_name}/{image}")
        for image in val_images:
          shutil.copy(path + f"/{class_name}/{image}", f"data/validation/{class_name}/{image}")
        for image in test_images:
          shutil.copy(path + f"/{class_name}/{image}", f"data/test/{class_name}/{image}")
```

سپس مدل خود را به صورت sequential تعریف می‌کنیم. به دلیل مسئله‌ی کلاس‌بندی با یک لیبل از تابع بهینه‌ساز adam و تابع ضرر categorical_crossentropy برای کامپایل مدل استفاده می‌کنیم.

```
model = Sequential([
                    layers.Conv2D(16,3,activation="relu",input_shape=(IMG_SHAPE,IMG_SHAPE,3)),
                    layers.MaxPooling2D(2),
                    layers.Conv2D(32,3,activation="relu"),
                    layers.MaxPooling2D(2),
                    layers.Conv2D(64,3,activation="relu"),
                    layers.MaxPooling2D(2),
                    layers.Flatten(),
                    layers.Dense(32,activation="relu"),
                    layers.Dense(5,activation="softmax")
])

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

print(model.summary())
```

همچنین call back ها را تعریف می‌کنیم.

call back اول برای زودتر متوقف کردن مدل در صورت عدم بهبود در داده‌های اعتبارسنجی است. دومی برای کنترل روند کاهش نرخ آموزش و سومی برای ذخیره‌ی بهترین مدل است.

```
early_stopping = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3)

reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=0)

checkpoint_filepath = '/content/checkpoints'

model_checkpoint = tf.keras.callbacks.ModelCheckpoint(
      filepath=checkpoint_filepath,
      save_weights_only=True)

callbacks = [
      early_stopping,
      reduce_lr,
      model_checkpoint]
```

سپس از دایرکتوری‌های مربوطه، مجموعه‌ی آموزشی و اعتبارسنجی را می‌سازیم. بدین منظور از Image Data Generator نیز برای انجام data augmentation استفاده می‌کنیم.

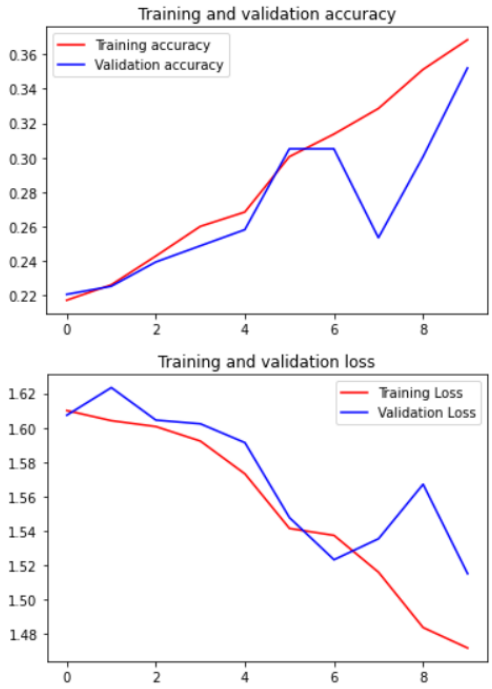
```
train_datagen = ImageDataGenerator(
  width_shift_range=0.1,
  height_shift_range=0.1,
  rescale=1./255,
  rotation_range=20,
  shear_range=0.2,
  zoom_range=0.2,
  horizontal_flip=True)

train_iterator = train_datagen.flow_from_directory(
  f"{BASE_FOLDER}/train/",
  batch_size=BATCH_SIZE,
  target_size=(IMG_SHAPE, IMG_SHAPE),
  shuffle=True,
  class_mode='categorical',
)

validation_datagen = ImageDataGenerator(rescale=1./255)

validation_iterator = validation_datagen.flow_from_directory(
  f"{BASE_FOLDER}/test/",
  batch_size=BATCH_SIZE,
  target_size=(IMG_SHAPE, IMG_SHAPE),
  class_mode='categorical',
)
```

سپس مدل را فیت می‌کنیم.



همانطور که می‌بینیم مدل دقت خوبی ندارد و دقت داده‌های اعتبارسنجی تقریبا به ۳۵٪ رسیده است.

بار دیگر از شبکه‌ی MobileNet استفاده می‌کنیم و مدل خود را تعریف می‌کنیم.

```
MobileNetV2 = tf.keras.applications.MobileNetV2(input_shape=(224, 224, 3), include_top=False, weights='imagenet',

model = Sequential()
model.add(MobileNetV2)
model.add(layers.Flatten())
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(5, activation='softmax'))


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

لایه‌ی آخر را false کرده و دو لایه‌ی dense به آن اضافه می‌کنیم. زیرا مسئله‌ی ما ۵ کلاسه است.

سپس مدل خود را کامپایل و فیت می‌کنیم.

همانطور که می‌بینیم مدل به دقت خوب ۸۰٪ در داده‌های آموزشی رسیده است اما دقت خوبی در داده‌های اعتبارسنجی ندارد و overfit شده است. بدین منظور می‌توان از روش‌های جلوگیری از overfitting استفاده کرد.