

:section1 توضیح

در این قسمت یک بار فقط با استفاده از کتابخانه `numpy` و یک بار هم به کمک `keras` عملیات `conv` را انجام داده ایم. خروجی اولی یک ماتریس 3×3 هست. خروجی دومی همان اعداد است اما صرفا `shape` آن فرق دارد و $1 \times 3 \times 3 \times 1$ است. دلیل سایز خروجی دومی این هست که تصویر ورودی عمقش یک است و تعداد فیلتر های خروجی نیز یک است.

:section2 توضیح

• توضیح :BasicBlock

در این قسمت از 2 لایه `conv` با سایز 3×3 استفاده شده که اولی دارای `stride` و دومی بدون `stride` است. بعد از هر `conv`, از تابع فعال سازی `ReLU` استفاده شده که روی داده های نرمال شده خروجی اعمال میشود. دقت کنید که اینجا `padding` باید انجام شود چرا که میخواهیم برای اینکه جمع با ورودی درست اتفاق بی افتد، عرض و ارتفاع ورودی یکسان باشد.

• توضیح :ResNet

در این قسمت معماری شبکه را پیاده کردیم. شبکه ای که ساخته ایم شامل یک 7×7 `conv` با 64 فیلتر، 3×3 `max pool` با `stride=2` و سپس شامل 8 `basic block` که هر جفت از آن ها به ترتیب 64 و 128 و 256 و 512 خروجی دارند. در انتها برای تبدیل کاهش پارامتر مدل از `dense` استفاده کردیم و سپس از لایه `global avgpool`

• توضیح قسمت `augmentation`: به این شکل است که یک تصویر را حداقل 50 درجه چرخانده ایم در جهت افقی و شیفت عکس 10 درصد است.

اموزش این شبکه بسیار زمان بر بود و متأسفانه برای بار اول که 10 تکرار بود الگوریتم ما، در epoch ششم نت قطع شد و اموزش مدل تقریباً بعد 5 ساعت متوقف شد. نتیجه اموزش مدل با تکرار 3 به این گونه هست ولی نتایج به دلیل کم بودن تکرار بالا نیست:

```

DIP_CNNs.ipynb ☆ Save failed
File Edit View Insert Runtime Tools Help
Commands + Code + Text
model.compile(optimizer = Adam(learning_rate=0.001),
              loss = 'categorical_crossentropy',
              metrics = ['accuracy'])

# Train the model with your own epochs
history = model.fit(
    train_generator,
    validation_data=val_generator,
    epochs=3,
    shuffle=True,
    verbose=1
)

/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call self._warn_if_super_not_called()
Epoch 1/3
303/303 ━━━━━━ 3090s 10s/step - accuracy: 0.1743 - loss: 2.4444 - val_accuracy: 0.1453 - val_loss: 3.6725
Epoch 2/3
303/303 ━━━━━━ 3053s 10s/step - accuracy: 0.2134 - loss: 2.1989 - val_accuracy: 0.1014 - val_loss: 17.9303
Epoch 3/3
303/303 ━━━━━━ 3038s 10s/step - accuracy: 0.2134 - loss: 2.1707 - val_accuracy: 0.1554 - val_loss: 3.8639

```

Automatic saving failed. This file was updated remotely or in another tab. Show diff

توضیح section3

در این قسمت ابتدا به کمک روش self supervised learning، از روی داده هایی که برچسب ندارند، یک خروجی که چقدر تصویر چرخیده تولید کرده ایم و سپس به کمک مدل imagenet آن را اموزش داده. نتایج به این شکل است:

The screenshot shows a Jupyter Notebook interface with the following details:

- Title Bar:** DIP_CNNs.ipynb, Save failed
- Toolbar:** File, Edit, View, Insert, Runtime, Tools, Help
- Code Cell:** # Train SSL model
ssl_model.fit(unlabeled_dataset, epochs=10)
Output: Epoch 1/10 to Epoch 10/10 progress bar and metrics.
- Variables Tab:** Shows a search bar and a list of variables.
- System Tray:** RAM: 16GB, Disk: 1TB, 2:08 PM, Python 3, 2:17 PM, 6/5/2025

سپس برای اموزش داده های با لیبل، از تکنیک **transfer learning** استفاده کرده ایم. به این گونه که از وزن های یادگرفته شده مدل قبلی استفاده کرده ایم. در این قسمت ما هر 3 حالت گفته شده را امتحان کردیم و چیزی که در توت بوک اورده شده است برای حالتی است که تمام وزن ها به جز لایه آخر **freez** شده و لایه آخر **train** شده اما با مقدار اولیه لایه آخر مدل قبلی. در هر 3 حالت خیلی نتایج جالب نبوده و با استفاده از این تکنیک به نتایج زیر رسیده ایم:

DIP_CNNs.ipynb Save failed

File Edit View Insert Runtime Tools Help

Commands + Code + Text RAM Disk

Epoch 1/13
4/4 20s 3s/step - accuracy: 0.2346 - loss: 1.9017 - val_accuracy: 0.2500 - val_loss: 1.8343
Epoch 2/13
4/4 18s 2s/step - accuracy: 0.3038 - loss: 1.8199 - val_accuracy: 0.1786 - val_loss: 1.7656
Epoch 3/13
4/4 10s 2s/step - accuracy: 0.2574 - loss: 1.8582 - val_accuracy: 0.2143 - val_loss: 1.8098
Epoch 4/13
4/4 7s 2s/step - accuracy: 0.2087 - loss: 1.8047 - val_accuracy: 0.1429 - val_loss: 1.6205
Epoch 5/13
4/4 8s 2s/step - accuracy: 0.2329 - loss: 1.7571 - val_accuracy: 0.2143 - val_loss: 1.7667
Epoch 6/13
4/4 7s 2s/step - accuracy: 0.3355 - loss: 1.6173 - val_accuracy: 0.3214 - val_loss: 1.4465
Epoch 7/13
4/4 9s 2s/step - accuracy: 0.3298 - loss: 1.6424 - val_accuracy: 0.4286 - val_loss: 1.5572
Epoch 8/13
4/4 9s 2s/step - accuracy: 0.2218 - loss: 1.7086 - val_accuracy: 0.3214 - val_loss: 1.6026
Epoch 9/13
4/4 7s 2s/step - accuracy: 0.2695 - loss: 1.6505 - val_accuracy: 0.1786 - val_loss: 1.6014
Epoch 10/13
4/4 10s 3s/step - accuracy: 0.1998 - loss: 1.6894 - val_accuracy: 0.2143 - val_loss: 1.6450
Epoch 11/13
4/4 18s 2s/step - accuracy: 0.2462 - loss: 1.7349 - val_accuracy: 0.2857 - val_loss: 1.6447
Epoch 12/13
4/4 9s 2s/step - accuracy: 0.3033 - loss: 1.6293 - val_accuracy: 0.3571 - val_loss: 1.5472
Epoch 13/13

Variables 2:08PM Python 3

Type here to search

