مدل‌سازی فرآیند بهینه‌سازی توپولوژی دو بعدی با استفاده از یادگیری عمیق

ابوالفضل یعقوبی

دانشجوی کارشناسی ارشد، دانشگاه صنعتی شریف، دانشکده مهندسی مکانیک، شماره دانشجویی: ۴۰۱۲۰۴۵۲۴، ایمیل: Abolfazl.dmg@gmail.com

چکیده

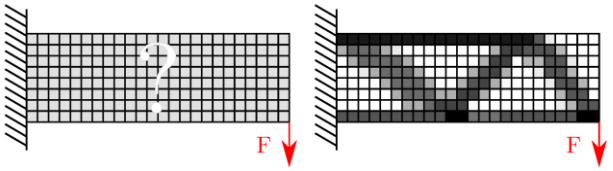
فرآیند بهینه‌سازی توپولوژی، فرآیندی مهم اما زمان‌گیر است. با استفاده از یادگیری ماشین، می‌توان این فرآیند را تا حد زیادی تسریع کرد. به این منظور در این پروژه مدل یادگیری عمیق به منظور پیش‌بینی نتیجه‌ی بهینه‌سازی توپولوژی با داشتن ورودی‌های مختلف مسئله، توسعه داده شده است. به منظور آموزش مدل، تعداد مناسبی داده با استفاده از الگوریتم بهینه‌سازی توپولوژی آماده شده است. ساختار شبکه‌ی عصبی با توجه به دقت بدست آمده انتخاب شده و نتایج بدست آمده پس از آموزش مدل، گزارش شده است.

واژه های کلیدی

بهینه‌سازی توپولوژی، یادگیری عمیق، شبکه عصبی

مقدمه

بهینه‌سازی توپولوژی از مهمترین مسائل مطرح در مهندسی است که امروزه توجه زیادی را به خود جلب کرده است. در بهینه‌سازی توپولوژی، توزیع ماده در فضای طراحی به گونه‌ای طراحی می‌شود که تابع هدف (که معمولا انرژی کرنشی جسم است) در حالت بهینه بوده و شروط اعمالی به مسئله ارضاء شوند. نمونه‌ای از مسئله‌ی بهینه‌سازی توپولوژی و نتیجه‌ی بدست آمده در شکل ۱ قابل مشاهده است.



شکل ۱: مسئله‌ی تیر کنتیلور و نتیجه بهینه‌سازی توپولوژی

مرور ادبیات

در سال ۲۰۰۴، بنزوئی و سیگموند [۱] اصول کلی بهینه‌سازی توپولوژی با استفاده از روش SIMP[۱] را در کتاب خود شرح دادند. ایشان همچنین الگوریتم و کد مربوط به این فرآیند را ارائه دادند که این الگوریتم با توسعه‌ی روش‌ها و تکنیک‌های جدید کد نویسی در طی سال‌ها دستخوش تغییر و پیشرفت زیادی شده است. سیگموند [۲] در سال ۲۰۲۰ کد مرجعی برای بهینه‌سازی توپولوژی ارائه داد که در آن از روش‌های به روز کدنویسی استفاده شده و سرعت اجرای کد تا حد زیادی بهبود یافته است.

شرح مسئله

با وجود پیشرفت‌هایی که در فرآیند بهینه‌سازی توپولوژی صورت گرفته، این فرآیند همچنان بسیار زمان‌گیر است. با توجه به قابلیت بالای یادگیری ماشین در سرعت بخشیدن به فرآیندهای مختلف و دقت بالای آن، می‌توان فرآیند بهینه‌سازی توپولوژی را با استفاده از آن تسریع کرد.

به این منظور، مسئله‌ی بهینه‌سازی دو بعدی با شبکه‌ی اجزای محدود $n \times m$ المان در نظر گرفته شده است. ورودی این مسئله، شرایط مرزی، بارگذاری اعمالی، قیود مورد انتظار همچون نسبت حجم و دیگر پارامترهای بهینه‌سازی توپولوژی هستند و خروجی، توزیع بهینه‌ی ماده در فضای طراحی است که به صورت ماتریسی $n \times m$ ارائه می‌شود. مولفه‌های این ماتریس، چگالی آن المان هستند که می‌توانند مقادیر صفر (نشان دهنده‌ی عدم وجود ماده) و یک (نشان دهنده‌ی ماده) را داشته باشند.

با این توضیح می‌توان این مسئله را به صورت یک مسئله‌ی دسته‌بندی دوتایی[۲] در نظر گرفت. برای حل این مسئله، از مدل یادگیری عمیق استفاده می‌شود. ورودی‌های این مدل می‌توانند شامل ویژگی‌های اصلی مسئله باشند که چند مورد از این ویژگی‌ها عبارتند از:

- جابجایی گره‌ها (در راستاهای افقی، عمودی و جابجایی مطلق)
- انرژی کرنشی ایجاد شده در المان‌ها در اثر بارگذاری
- تنش ایجاد شده در المان‌ها در اثر بارگذاری
- ضریب پنالتی استفاده شده در فرآیند SIMP
- قیود اعمالی به مسئله از جمله نسبت حجم

[۱] Solid Isotropic Material with Penalization

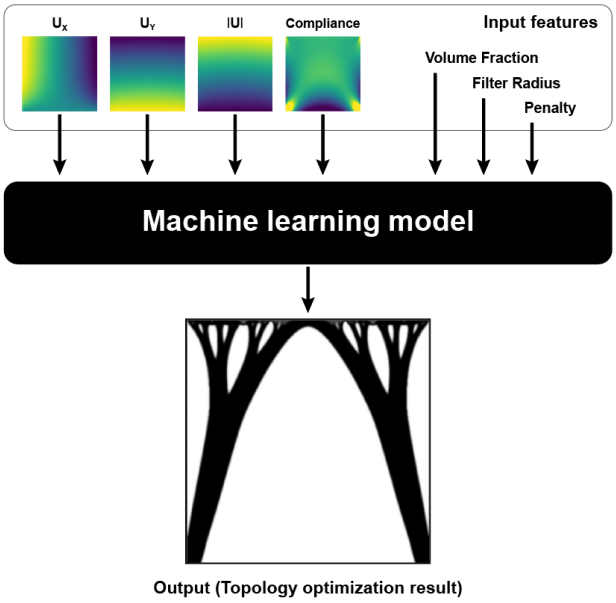
[۲] Binary classification

- شعاع فیلتر اعمالی بر روی متغیر طراحی
- دیگر پارامترهای بهینه‌سازی توپولوژی

در این پروژه، هفت ویژگی از موارد فوق برای مدل‌سازی انتخاب شده‌اند. این ویژگی‌ها عبارتند از:

- جابجایی گره‌ها (در راستاهای افقی، عمودی و جابجایی مطلق)
- انرژی کرنشی ایجاد شده در المان‌ها در اثر بارگذاری
- ضریب پنالتی استفاده شده در فرآیند SIMP
- قیود اعمالی به مسئله از جمله نسبت حجم
- شعاع فیلتر اعمالی بر روی متغیر طراحی

بنابراین شماتیک کلی از مدل یادگیری عمیق مورد استفاده در این پژوهش به صورت شکل ۲ است.

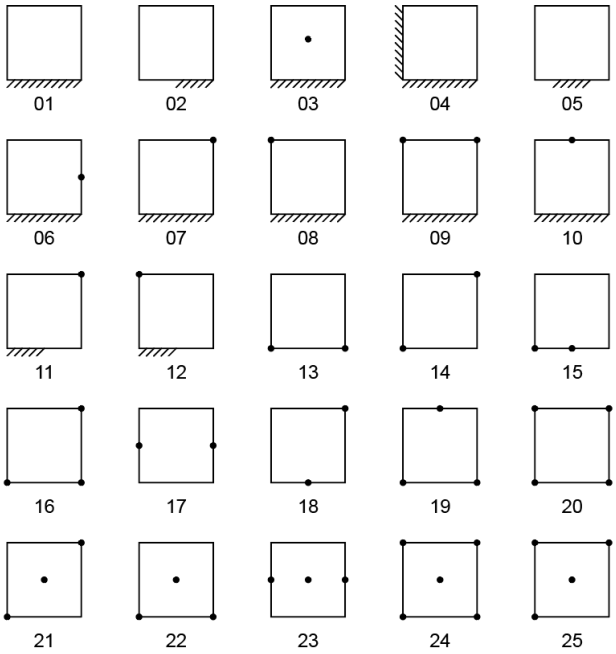


شکل ۲: شماتیک کلی از مسئله یادگیری ماشین

آماده سازی داده‌ها

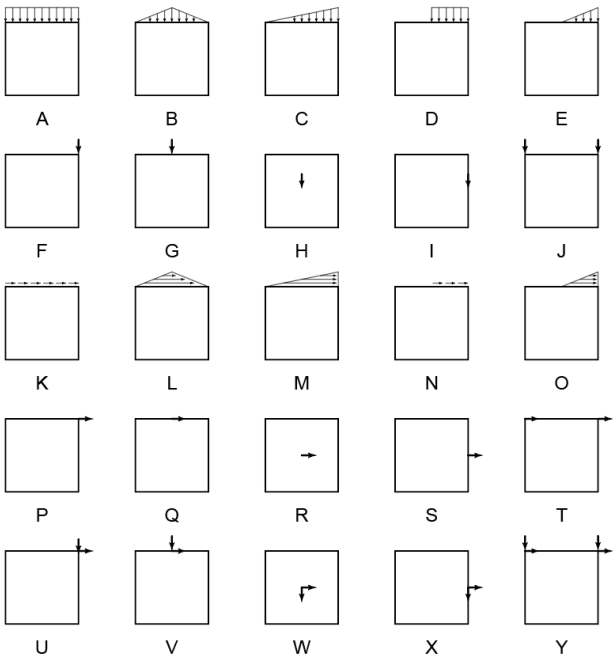
آموزش مدل یادگیری ماشین، نیاز به تعداد مناسبی داده دارد که حالت‌های متنوعی از ورودی و خروجی را شامل شود. به منظور آماده سازی داده‌های آموزشی، از الگوریتم ارائه شده توسط سیگموند [2] استفاده شده است. این الگوریتم به شکل بهینه در فایل "**TopOpt.py**" پیاده سازی شده است. این الگوریتم به منظور بهینه‌سازی توپولوژی **۲۵۰،۰۰۰** حالت مختلف استفاده شده که شامل ترکیب ۲۵ شرایط مرزی مختلف، ۲۵ بارگذاری مختلف، ۴ ضریب پنالتی مختلف، ۱۰ شعاع اعمال فیلتر مختلف و ۱۰ نسبت حجم مختلف است.

شکل ۳، حالات مختلف شرایط مرزی در نظر گرفته شده به همراه کد اختصاص داده شده به هر یک را نشان می‌دهد.



شکل ۳: حالات مختلف در نظر گرفته شده برای شرایط مرزی

همچنین حالات مختلف بارگذاری به همراه کد اختصاص داده شده به هر یک در شکل ۴ قابل مشاهده است.



شکل ۴: حالات مختلف در نظر گرفته شده برای بارگذاری

لازم به ذکر است که برخی از شرایط مرزی با برخی از بارگذاری‌ها سازگاری ندارند (مانند شرایط مرزی 20 و بارگذاری F). این مشکل در بخش پیش‌پرازش داده‌ها و با اعمال فیلتر برای انتخاب داده‌های مناسب برطرف شده است.

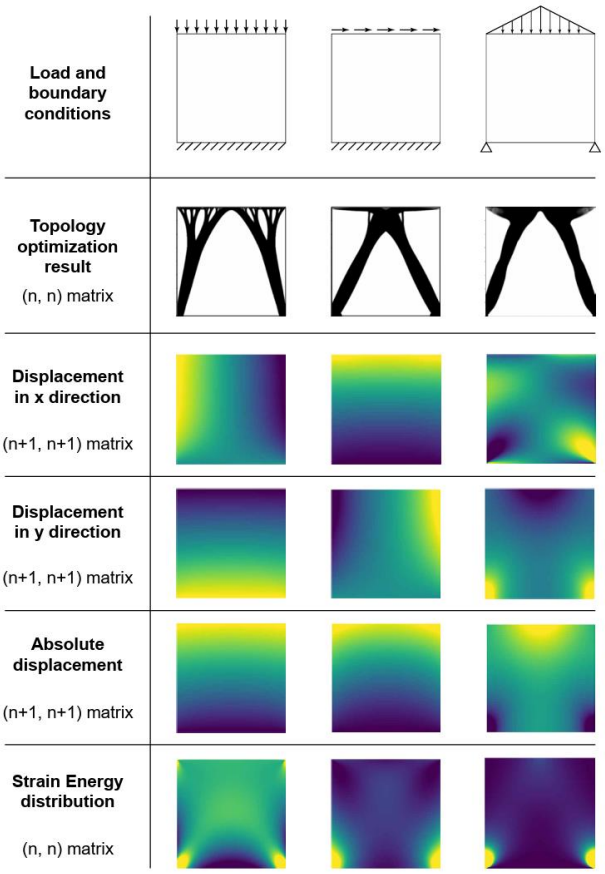
ضرایب پنالتی در نظر گرفته شده، ۳، ۴، ۵ و ۶ هستند. همچنین مقادیر مختلف شعاع اعمال فیلتر عبارتند از:
1.75, 2.00, 2.25, 2.50, 2.75, 3.00, 4.00, 5.00, 6.00, 8.00

و مقادیر در نظر گرفته شده برای نسبت حجم عبارتند از:
0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55, 0.6, 0.65

تمامی این حالات در فایل "**modes.py**" تعریف شده‌اند.
اطلاعات ذخیره شده در هر اجرا عبارتند از:

- ضریب پنالتی (عدد حقیقی مثبت)
- شعاع اعمال فیلتر (عدد حقیقی مثبت)
- نسبت حجم (عدد حقیقی بین ۰ و ۱)
- جابجایی در راستای x (ماتریس $(n+1) \times (n+1)$)
- جابجایی در راستای y (ماتریس $(n+1) \times (n+1)$)
- جابجایی مطلق (ماتریس $(n+1) \times (n+1)$)
- انرژی کرنشی (ماتریس $n \times n$)
- نتیجه‌ی بهینه‌سازی (ماتریس $n \times n$)

شکل ۵ داده‌های ماتریسی ذخیره شده در هر اجرا برای چند حالت مختلف را نشان می‌دهد.

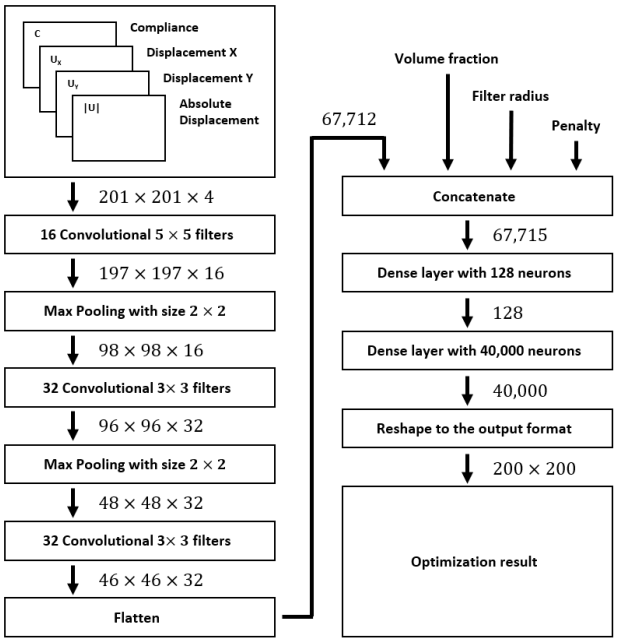


شکل ۵: مقادیر ماتریسی ذخیره شده در هر اجرا برای سه حالت مختلف

فضای طراحی در مسئله، به صورت مربعی با تعداد المان **۲۰۰×۲۰۰** در نظر گرفته شده است. فرآیند تولید داده به صورت خودکار و با بهره‌گیری از سرویس گوگل کولب[۳] با استفاده از کد نوشته شده در فایل "**prepare_data.py**" انجام شده است. داده‌های بدست آمده، در پوشه‌ی "**data**" ذخیره شده‌اند. به دلیل حجم بالای این داده‌ها (حدود ۵۰ گیگابایت)، تنها بخشی از آن به همراه این گزارش ارائه شده است اما در صورت نیاز تمامی داده‌ها نیز قابل ارائه هستند.

معماری شبکه

معماری‌های مختلفی برای شبکه‌ی عصبی یادگیری عمیق طراحی شده و مورد آزمایش قرار گرفته است. یک مورد از این معماری‌ها در شکل ۶ قابل مشاهده است.



شکل ۶: معماری شبکه با بهره گیری از سرویس تابعی کراس[۴]

در این معماری، ورودی‌های دو بعدی مسئله در ابتدا از چندین لایه فیلترهای پیچشی[۵] عبور کرده و پس از خطی سازی، با ورودی‌های عددی ترکیب شده و وارد شبکه عصبی تمام متصل[۶] می‌شوند. در انتها، خروجی به شکل ماتریس در می‌آید.
پیاده‌سازی این شبکه، با کمک کتابخانه‌های TensorFlow و Keras انجام شده است. کد شکل ۷ مربوط به این مدل است.

[۳] Google Colab
[۴] Keras functional API
[۵] Convolutional filters
[۶] Fully connected Neural Network

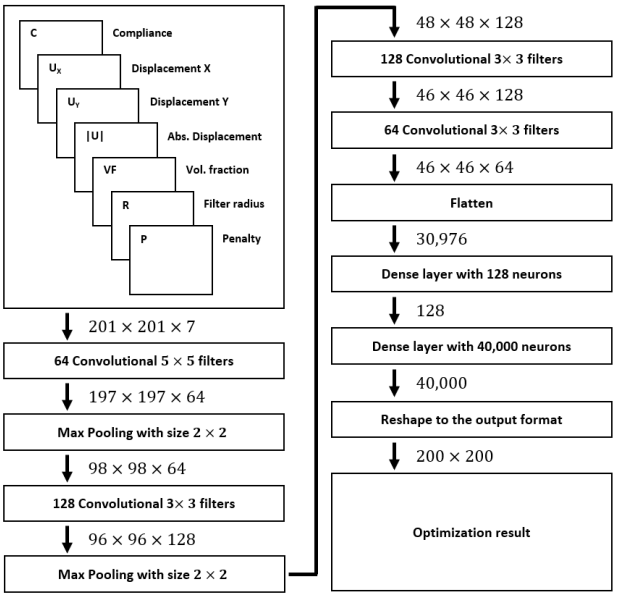
```
input_layer_1d = Input(shape=(3, ))
input_layer_2d = Input(shape=(201, 201, 4))

conv1 = Conv2D(16, kernel_size=(5, 5), activation='relu')(input_layer_2d)
pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)
conv2 = Conv2D(32, kernel_size=(3, 3), activation='relu')(pool1)
pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)
conv3 = Conv2D(32, kernel_size=(3, 3), activation='relu')(pool2)
flat1 = Flatten()(conv3)
merge = Concatenate()([flat1, input_layer_1d])
flat2 = Flatten()(merge)
dense1 = Dense(64, activation='relu')(flat2)
dense2 = Dense(128, activation='relu')(dense1)
dense3 = Dense(200 * 200, activation='sigmoid')(dense2)
output = Reshape((200, 200))(dense3)

model = Model(inputs=[input_layer_1d, input_layer_2d], outputs=output)
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

شکل ۷: کد مربوط به پیاده‌سازی معماری شبکه

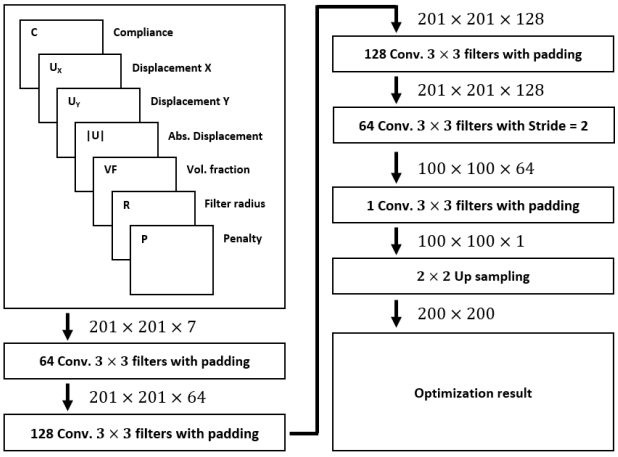
همانطور که اشاره شد، مسئله از نوع دسته‌بندی است. به همین دلیل از تابع زیان Binary Cross Entropy استفاده شده است. متاسفانه این معماری پس از ۱۰ اپوک[۷] آموزش روی داده‌ها، دقتی حدود ۴۰ درصد بدست داد که دقت بسیار پایینی است و نشان از نامناسب بودن این معماری برای این مسئله دارد.
دومین معماری طراحی شده در شکل ۸ قابل مشاهده است.



شکل ۸: معماری شبکه با تعریف ورودی‌های عددی بصورت ماتریس

در این معماری، ورودی‌های عددی نیز به صورت ماتریس‌هایی دو بعدی به شبکه داده شده‌اند که تمامی مولفه‌های این ماتریس‌ها ثابت هستند. با این کار می‌توان تاثیر این پارامترها را نیز در اعمال فیلترهای پیچشی اعمال کرد. این معماری نیز پس از ۱۰ اپوک آموزش روی داده‌ها، دقتی حدود ۵۵ درصد را بدست داد که هرچند نسبت معماری قبل دقت بهتری است، اما همچنان بسیار پایین است و مناسب نیست.

معماری انتخاب شده برای مسئله، در شکل ۹ قابل مشاهده است.



شکل ۹: معماری شبکه‌ی انتخاب شده

در این معماری نیز تمام ورودی‌ها به صورت ماتریسی به شبکه داده شده‌اند اما بخش تمام متصل شبکه حذف شده و تنها از فیلترهای پیچشی استفاده شده است. پیاده‌سازی این شبکه در شکل ۱۰ نشان داده شده است.

```
model = Sequential()
model.add(Conv2D(64, kernel_size=(3, 3), padding='same', activation='relu',
input_shape=(201, 201, 7)))
model.add(Conv2D(128, kernel_size=(3, 3), padding='same', activation='relu'))
model.add(Conv2D(128, kernel_size=(3, 3), padding='same', activation='relu'))
model.add(Conv2D(64, kernel_size=(3, 3), strides=2, activation='relu'))
model.add(Conv2D(1, kernel_size=(3, 3), padding='same', activation='sigmoid'))
model.add(UpSampling2D(size=(2, 2)))

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

شکل ۱۰: کد مربوط به پیاده‌سازی معماری شبکه‌ی انتخاب شده

این شبکه با اینکه نسبت به معماری‌های قبلی ساده تر است اما دقت خیلی خوب ۹۸ درصد را پس از آموزش روی داده‌ها بدست می‌دهد و انتخاب مناسبی برای مسئله است.

پیش‌پردازش داده‌ها

به منظور استفاده‌ی داده‌ها در آموزش مدل، لازم است این داده‌ها در رم بارگذاری شوند. اما به دلیل حجم زیاد داده‌ها، این کار امکان پذیر نیست. به همین دلیل از بستر تنسورفلو دیتاست[۸] استفاده شده است. این بستر امکانات زیادی از جمله امکان بارگذاری داده‌ها به صورت تدریجی، امکان اعمال فیلتر روی داده‌ها و عدم استفاده از داده‌های نامناسب، امکان چینش تصادفی داده‌ها و امکان تقسیم داده‌ها به بچ[۹] با سایز موردنظر را فراهم می‌کند. در ادامه مراحل طی شده در پیش‌پردازش داده‌ها ارائه شده است.

---

[۷] Epoch
[۸] TensorFlow.Data.Dataset
[۹] Batch

تعریف دیتاست

دیتاست با لحاظ کردن تمامی حالات مختلف داده‌ها تعریف شده و به زیرمجموعه‌های آموزش، اعتبارسنجی و آزمایش تقسیم شده است. این کار با استفاده از دستورات شکل ۱۱ انجام شده است.

```
parameters = [(b, l, p, r, v) for b in list(bcs.keys()) for l in list(loads.keys())
              for p in ps for r in rs for v in vfs]

train_val_params, test_params = train_test_split(parameters, test_size=0.1, shuffle=True)
train_params, validation_params = train_test_split(train_val_params, test_size=0.1, shuffle=True)

train_params = tuple(np.array(item) for item in zip(*train_params))
validation_params = tuple(np.array(item) for item in zip(*validation_params))
test_params = tuple(np.array(item) for item in zip(*test_params))

train_ds = tf.data.Dataset.from_tensor_slices(train_params)
validation_ds = tf.data.Dataset.from_tensor_slices(validation_params)
test_ds = tf.data.Dataset.from_tensor_slices(test_params)
```

شکل ۱۱: کد مربوط به تعریف دیتاست

بارگذاری داده‌ها

هر داده لازم است ابتدا از حافظه خوانده شده سپس به فرمت مشخص ورودی و خروجی درآید. تابع load_data به همین منظور پیاده سازی شده است. در این تابع، برای هر حالت مشخص از شرایط مرزی، بارگذاری، ضریب پنالتی، شعاع فیلتر و نسبت حجم، داده‌ها از حافظه خوانده می‌شوند. سپس این داده‌ها نرمالایز شده و داخل متغیر x با ابعاد ۷×۲۰۱×۲۰۱ ذخیره می‌شوند. لازم به ذکر است که چون ابعاد ماتریس انرژی کرنشی ۲۰۰×۲۰۰ است، این ماتریس ابتدا گسترش داده شده است.

خروجی مسئله، به صورت ماتریس دودوئی[۱۰] با ابعاد ۲۰۰×۲۰۰ است. به همین دلیلی مقداری مشخص به عنوان آستانه[۱۱] در نظر گرفته شده که مقادیر پیوسته‌ی ماتریس خروجی را ناپیوسته می‌کند. شکل ۱۲ کد مربوط به این بخش را نشان می‌دهد.

```
def load_data(bc_code, load_code, penal, r, vf, threshold=0.3):
    bc_code, load_code, penal, r, vf = [param.numpy() for param in (bc_code, load_code, penal, r, vf)]
    bc_code, load_code = bc_code.decode('utf-8'), load_code.decode('utf-8')
    load_path = 'data/' + bc_code + '/' + load_code + '/'
    code = f'{bc_code}-{load_code}-P{penal}-R{r:0.3f}-V{vf:0.3f}'

    c = np.load(load_path + bc_code + load_code + '-C.npy')
    ux = np.load(load_path + bc_code + load_code + '-X.npy')
    uy = np.load(load_path + bc_code + load_code + '-Y.npy')

    x = np.ones((201, 201, 7), dtype=np.float32)
    x[:, :, 0] = MinMaxScaler().fit_transform(ux)
    x[:, :, 1] = MinMaxScaler().fit_transform(uy)
    x[:, :, 2] = MinMaxScaler().fit_transform(np.hypot(ux, uy))
    x[:, :, 3] = MinMaxScaler().fit_transform(np.pad(c, pad_width=1, mode='edge')[0:-1, 0:-1])
    x[:, :, 4], x[:, :, 5], x[:, :, 6] = penal, r, vf

    y = np.load(load_path + 'output/' + code + '.npy')
    y[y >= threshold] = 1
    y[y < threshold] = 0

    return x, y.astype(np.bool_)
```

شکل ۱۲: کد مربوط به بارگذاری داده‌ها

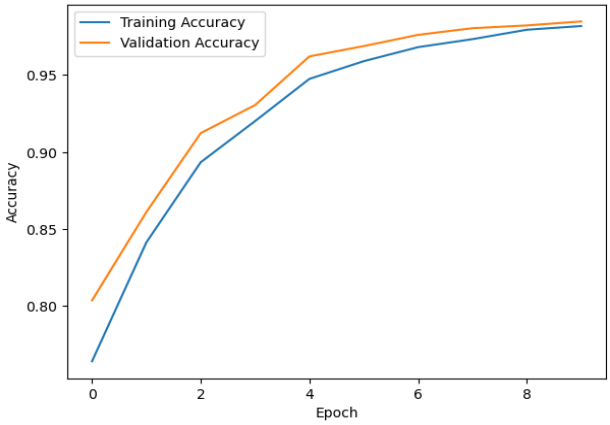
اعمال فیلتر بر روی داده‌ها

ممکن است برخی از داده‌ها در حین فرآیند تولید داده، به دلایلی به طور درست تولید نشده باشند. همچنین ممکن است برخی از داده‌ها مناسب فرآیند یادگیری نباشند که به یک مورد از آن‌ها در بخش آماده سازی داده‌ها اشاره شد.

به منظور اجتناب از بروز خطا لازم است داده‌های مناسب برای آموزش انتخاب شوند. این کار با استفاده از تابع filter_data انجام می‌شود. در این تابع ابتدا وجود داده بررسی شده و در صورت وجود داده، بررسی می‌شود که مقدار انرژی کرنشی صفر نباشد. داده‌هایی که از این دو فیلتر عبور کنند برای فرآیند یادگیری مناسب هستند.

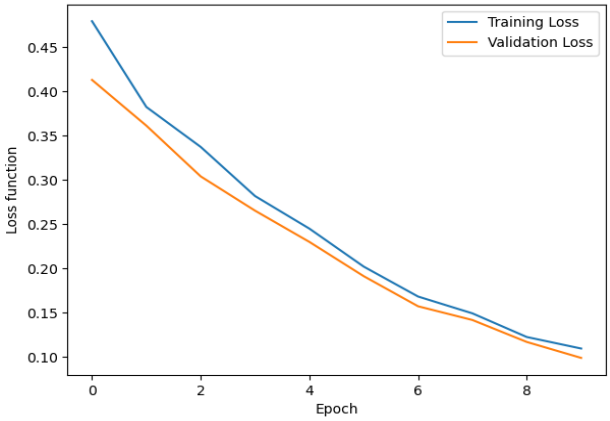
آموزش مدل

آموزش مدل با استفاده از داده‌های آموزش در ۱۰ اپوک انجام شده و دقت مدل در هر اپوک با استفاده از داده‌های اعتبارسنجی سنجیده شده است. شکل ۱۳ نمودار تغییرات دقت مدل در طول فرآیند آموزش را نشان می‌دهد.



شکل ۱۳: تغییرات دقت مدل در فرآیند آموزش

نمودار، همگرایی مدل و روند درست آموزش را نشان می‌دهد. در انتهای آموزش، دقت مدل روی داده‌های آموزش ۹۸.۱۸ درصد و روی داده‌های اعتبارسنجی ۹۸.۴۸ درصد است. تغییرات تابع هزینه در طول فرآیند آموزش نیز در شکل ۱۴ قابل مشاهده است.



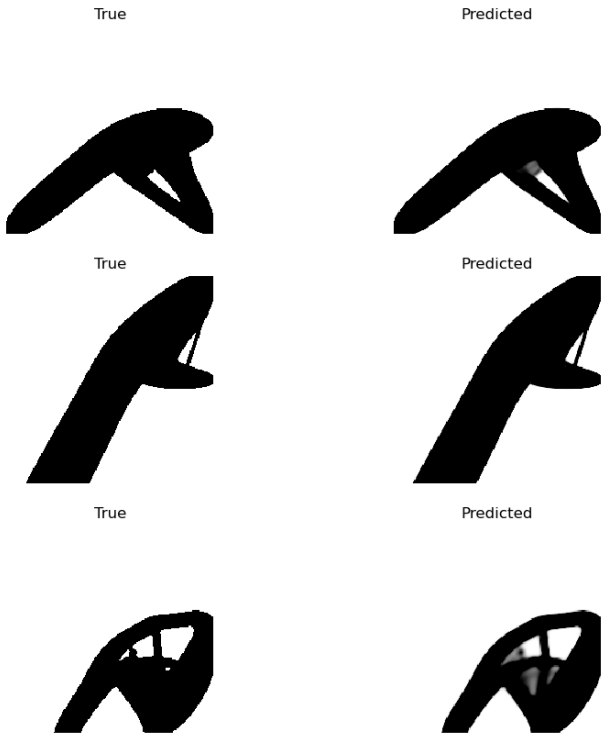
شکل ۱۴: تغییرات تابع هزینه در فرآیند آموزش

[۱۰] Binary

[۱۱] Threshold

نتایج

پس از آموزش، دقت مدل روی داده‌های آزمایش مورد بررسی قرار گرفته است. نتیجه‌ی این بررسی نشان می‌دهد که مدل دارای دقت **۹۸.۳۸ درصد** بوده و مقدار تابع هزینه برابر ۰.۱۱۰۶ است. این دقت، بسیار مناسب است و نشان دهنده‌ی کارآیی بالای مدل است. برای نمایش توانایی مدل در پیش‌بینی نتیجه‌ی بهینه‌سازی توپولوژی، سه مورد از داده‌های آزمایش به صورت تصادفی انتخاب شده و در شکل ۱۵ نشان داده شده‌اند.



شکل ۱۵: مقایسه‌ی نتایج واقعی و پیش‌بینی شده

جمع بندی

در این پروژه، از یادگیری ماشین برای مدل‌سازی بهینه‌سازی توپولوژی و سرعت بخشیدن به آن استفاده شد. معماری‌های مختلفی برای شبکه عصبی طراحی شده و بهترین معماری با توجه به قابلیت پیش‌بینی انتخاب شد. سپس مدل طراحی شده با استفاده از داده‌های تولید شده با استفاده از الگوریتم بهینه‌سازی توپولوژی، آموزش داده شد. در هر مرحله آموزش، دقت مدل با استفاده از داده‌های اعتبارسنجی سنجیده شده و هایپرپارامترها تصحیح شدند. پس از فرآیند آموزش، دقت مدل بر روی داده‌های آزمایش مورد بررسی قرار گرفت. نتایج بدست آمده نشان از دقت و توانایی بالای مدل دارند. به طوری که دقت مدل در پیش‌بینی داده‌های آزمایشی، ۹۸.۳۸ درصد است.

نتایج بدست آمده در این پژوهش، توانایی بالای یادگیری ماشین را نشان می‌دهند. با توجه به قابلیت‌های بالای این تکنولوژی، پیش‌بینی می‌شود در آینده بسیاری از فرآیندهای زمان‌گیر با بهره‌گیری از این تکنولوژی تسریع شوند.

مراجع

[1] Bendsøe, M. P., & Sigmund, O. (2004b). Topology Optimization. In Springer eBooks. https://doi.org/10.1007/978-3-662-05086-6

[2] Ferrari, F., & Sigmund, O. (2020). A new generation 99 line MATLAB code for compliance topology optimization and its extension to 3D. Structural and Multidisciplinary Optimization, 62(4), 2211–2228. https://doi.org/10.1007/s00158-020-02629-w