گزارش پروژه جستجوی معماری عصبی

کیارش صدقی قادیکلائی
دانشکده مهندسی کامپیوتر
دانشگاه اصفهان
kiarash.sedghi99@gmail.com

چکیده

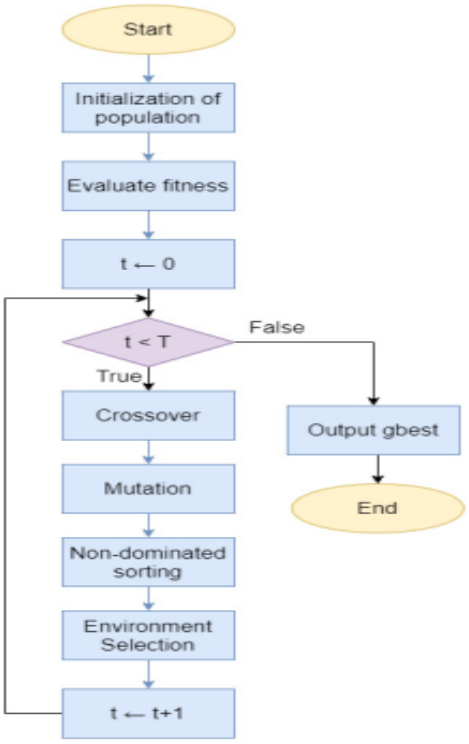
امروزه یادگیری عمیق توانسته است موفقیت چشمگیری در بسیاری از زمینه‌ها ازجمله تشخیص تصویر، تشخیص اشیاء و ترجمه ماشین به دست بیاورد و اثبات‌شده است که طراحی صحیح معماری‌های عصبی می‌تواند بر نمایش ویژگی‌ها و بهبود عملکرد نهایی تأثیر داشته باشد. با توسعه و پیشرفت یادگیری عمیق، طراحی یک ساختار شبکه عصبی مناسب به‌عنوان یک مسئله‌ی مهم و حیاتی محسوب می‌شود. طراحی یک شبکه به دانش قبلی متخصصان و تجربه‌ی آن‌ها بستگی دارد و همین موضوع سبب می‌شود، افرادی که تجربه‌ی کمی در این موضوع دارند، در ارائه پیشنهادهای مناسب دچار مشکل شوند. همچنین بدیهی است که دانش افراد متخصص درنهایت محدود به یکی دامنه خاص از مسائل خواهد شد و به‌مرور، منقضی خواهد شد. در سال‌های اخیر، موفقیت در استفاده از Neural Architecture Search (NAS) توانسته است در فرآیند طراحی یک شبکه، نیروهای متخصص انسانی را به‌تدریج با الگوریتم‌های کامپیوتری جایگزین نماید. الگوریتم و متدهای مختلفی برای NAS ارائه‌شده‌اند که به‌طورکلی مبتنی بر یادگیری تقویتی و یادگیری تکاملی می‌باشند. در این گزارش، به نقش یادگیری تکاملی در طراحی شبکه‌های عصبی پرداخته خواهد شد و برنامه‌ی kNas که پیاده‌سازی NAS مبتنی بر الگوریتم‌های تکاملی می‌باشد، معرفی خواهد شد.

۱ مقدمه

توسعه خودکار یک شبکه عصبی بدون دخالت نیروی انسانی متخصص موضوعی کاملاً جدید نمی‌باشد و از دهه‌های گذشته در مقالات [۴] و [۵] تا به الآن در [۶] و [۷] به آن پرداخته‌شده است. جدیدترین الگوریتم‌های معرفی‌شده در این حوزه به‌طورکلی مبتنی بر دو الگوریتم یادگیری تقویتی (RL) [۶] و [۷] و الگوریتم تکاملی (EA) [۸] و [۹] می‌باشند. در روش‌هایی که مبتنی بر یادگیری تقویتی هستند، انتخاب یا عدم انتخاب یک بخش از معماری به‌عنوان عمل (Action) در نظر گرفته می‌شود. درنهایت یک دنباله از عمل‌ها، معماری نهایی را مشخص می‌کند که با ارزیابی مدل با داده‌های ارزیابی، دقت مدل، مشخص‌کننده‌ی مناسب بودن دنباله‌ی عمل‌ها خواهد بود. در الگوریتم‌های مبتنی بر الگوریتم تکاملی، یک بخش از معماری، در فرآیند ترکیب و جهش راه‌حل‌های مسئله به دست می‌آید و بهترین معماری‌های به‌دست‌آمده در هر مرحله برای مرحله‌ی بعد انتخاب می‌شوند. با توجه به ماهیت مسئله و ازآنجایی‌که راه‌حل‌های مسئله راه‌حل‌های گسسته هستند، به این معنا که یک راه‌حل می‌تواند تنها در اندازه کرنل لایه‌های کانولوشنی با یک راه‌حل دیگر متفاوت باشد، تمامی متدهای معرفی‌شده برای هردو دسته از الگوریتم‌های قبل، جستجوی خود را در یک فضای گسسته انجام می‌دهند. بدون شک، ساده‌ترین روش برای رسیدن به پاسخ مسئله، جستجوی کامل فضای راه‌حل مسئله باشد. اما این راه‌حل، روش کارآمدی نیست چراکه فضای پاسخ می‌تواند به‌صورت نمایی رشد نماید. همین موضوع سبب می‌شود تا از روش‌های جستجوی بهینه و هوشمند استفاده شود. در این گزارش، در بخش دوم به کاربرد الگوریتم‌های تکاملی در یافتن پاسخ مسئله پرداخته خواهد شد. در بخش سوم مسئله جستجوی معماری در قالب یک مسئله بهینه‌سازی مطرح خواهد شد و الگوریتم‌های مناسب برای آن معرفی خواهد شد. در بخش چهارم، به معرفی برنامه‌ی kNas که به‌منظور پیاده‌سازی NAS مبتنی بر الگوریتم‌های تکاملی ارائه‌شده است، پرداخته خواهد شد. در بخش پنجم نیز، نتایج و تحلیل مسئله ارائه خواهد شد.

۲ الگوریتم‌های تکاملی

همان‌طور که می‌دانیم، یک الگوریتم تکاملی با ایجاد یک جمعیت اولیه از راه‌حل‌ها آغاز می‌شود. راه‌حل‌ها در این مسئله، بیانگر لایه‌های مختلف و پارامترهای مختص به هر لایه می‌باشد. به‌عنوان‌مثال، یک راه‌حل بیانگر موارد زیر است: یک شبکه با ۴ لایه کانولوشنی که تمامی لایه‌ها غیر از لایه دوم دارای تابع فعال‌سازی می‌باشند، تمامی لایه‌ها دارای max pooling می‌باشند و اندازه کرنل در تمامی لایه‌ها برابر ۴ می‌باشد. بعد از ایجاد جمعیت اولیه نوبت به ارزیابی آن‌ها می‌رسد. ارزیابی یک راه‌حل به معنای آموزش آن شبکه توسط داده‌های آموزشی از پیش تعیین‌شده و سپس ارزیابی آن توسط داده‌های ارزیابی می‌باشد. درنهایت دقت به‌دست‌آمده، بیانگر میزان برازندگی آن راه‌حل خواهد بود. مرحله بعد، تکرار فرآیند انتخاب راه‌حل‌ها از جمعیت، ترکیب آن‌ها، جهش راه‌حل‌های جدید تولیدشده و برگرداندن راه‌حل‌های جدید در صورت لزوم به جمعیت می‌باشد. فرآیندهای فوق در شکل ۱ نشان داده‌شده‌اند.



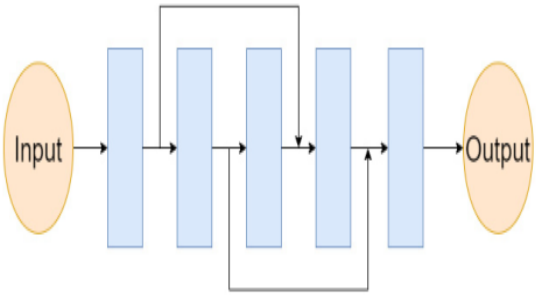
شکل ۱: چهارچوب الگوریتم تکاملی [۲]

در بخش بعدی به معرفی هر بخش از الگوریتم تکاملی و نحوه مربوط کردن آن‌ها به مسئله‌ی جستجوی معماری پرداخته خواهد شد.

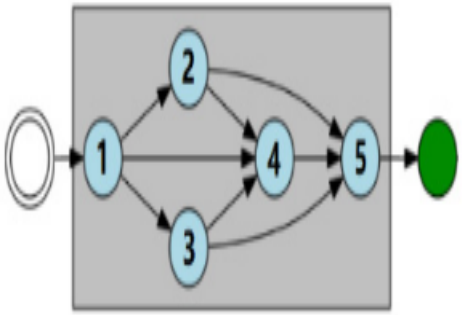
۳ الگوریتم تکاملی در جستجوی معماری

در بخش قبل، کلیت الگوریتم تکاملی معرفی گردید. در این بخش گام‌به‌گام، مسئله جستجوی معماری را در قالب یک مسئله‌ای که الگوریتم‌های تکاملی حل می‌کنند، بیان خواهیم کرد. اولین مرحله، مشخص نمودن نحوه‌ی نمایش راه‌حل‌های مسئله می‌باشد. همان‌طور که گفته شد، یک راه‌حل شامل مجموعه‌ای از لایه‌ها به دنبال هم می‌باشند که هر لایه در شبکه، شامل مجموعه‌ای از اجزا می‌باشد. به‌طورکلی، تمامی اجزای تمامی لایه‌ها از ورودی تا خروجی را می‌توان به دو صورت نمایش داد:

- ساختار خطی:در این ساختار، اجزاء به‌صورت خطی به دنبال هم قرارگرفته‌اند و می‌توان آن‌ها را به‌صورت خطی نمایش داد.
- ساختارگرافی: در این ساختار، اجزاء الزاماً به‌صورت خطی به دنبال هم قرار نگرفته‌اند. همین موضوع سبب می‌شود نمایش خطی دیگر کاربرد نداشته باشد. شکل (۲) به‌خوبی نشان می‌دهد که در برخی از موارد، نمایش خطی امکان‌پذیر نمی‌باشد و لازم است تا از ساختارگرافی شکل (۳) استفاده نمود.



شکل ۲: ساختار غیرخطی شبکه [۲]



شکل ۳: نمایش گرافی شبکه [۲]

در این پروژه تنها حالتی که لایه‌ها و اجزاء به‌صورت خطی کنار هم قرارگرفته‌اند موردبررسی قرارگرفته است.

در برنامه‌ی kNas هر راه‌حل توسط یک کلاس تحت عنوان KNasEAIndividual که در فایل knasEA.py پیاده‌سازی شده است، مشخص می‌شود. این کلاس شامل ویژگی مختلفی مانند لیست لایه‌های کانولوشنی موجود در شبکه، لایه چگال موجود در شبکه، مقدار برازندگی، نرخ یادگیری شبکه و اطلاعات مربوط به ارزیابی شبکه می‌باشد. تمام ویژگی‌های مطرح‌شده در این کلاس، یک راه‌حل را به‌خوبی توصیف می‌کنند. دقت شود که همان‌طور که معرفی شد، یک راه‌حل یک لیست از لایه‌ها را نگهداری می‌کند که این موضوع خود، نشان‌دهنده نحوه نمایش خطی است.

در برنامه‌ی kNas، به‌طورکلی لایه‌های شبکه به دو دسته‌ی لایه‌های کانولوشنی و چگال دسته‌بندی می‌شوند. این دو لایه به ترتیب در قالب کلاس‌های CNLayer و DFCLayer در فایل knasLN.py پیاده‌سازی شده‌اند. هر دو کلاس شامل ویژگی‌هایی ازجمله تعداد فیلترها، داشتن یا نداشتن batch norm، داشتن یا نداشتن تابع فعال‌سازی، داشتن یا نداشتن dropout و داشتن یا نداشتن max pooling می‌باشند. هردو کلاس نیز از متدهایی استفاده می‌کنند که می‌توانند به‌صورت تصادفی یا به‌طور دلخواه، یک لایه را ایجاد و به شبکه اضافه کنند. بعد از مشخص شدن نحوه نمایش راه‌حل‌ها در برنامه kNas، به مرحله بعد یعنی ارزیابی آن می‌رسیم. ارزیابی یک راه‌حل با کنار هم قرار دادن لایه‌های آن و آموزش و سپس ارزیابی آن توسط داده‌های ارزیابی امکان‌پذیر است. پیاده‌سازی ارزیابی مدل در فایل‌های knasLN.py که شکل کلی مدل را در قالب کلاس KNasLayersNet معرفی می‌کند و همچنین فایل knasModel.py که به تولید و ارزیابی مدل در قالب کلاس KNasModel و متد $knas\_create\_eval\_model()$ می‌پردازد، می‌باشد. خروجی ارزیابی، شامل مجموعه‌ای از اطلاعات شامل، مدت‌زمان آموزش هر epoch، میزان دقت و ضرر در داده‌های آموزشی، صحت‌سنجی و ارزیابی در هر epoch می‌باشد.

مقدار برازندگی هر راه‌حل توسط روابط زیر تعیین می‌شود:

$$P1 = meanTestAcc \tag{۱}$$

$$P2 = 1/(|meanValAcc - meanTestAcc| + 1) \tag{۲}$$

$$P3 = 1/meanTrainDuration \tag{۳}$$

$$P4 = 1/numLearnableParams \quad (۴)$$

$$fitness = P1 + P2 + P3 + P4 \quad (۵)$$

یکی از مواردی که می‌تواند عملکرد الگوریتم را بهبود بخشد، استفاده از fingerprint است. به این شکل که هر ترکیب جدیدی که تولید می‌شود، یک امضا یا fingerprint از آن نگهداری می‌شود. هرگاه که نیاز به مقدار برازندگی آن راه‌حل بود، به جای انجام محاسبات سنگین آموزش و ارزیابی، می‌توان مستقیم از مقدار قبلی آن استفاده نمود. این روش زمانی موثر است که در طول فرآیند تکامل، به راه‌حل‌هایی برخورد می‌کنیم که قبلا آن‌ها را بررسی کرده‌ایم. راه تشخیص این موضوع نیز نگهداری یک امضا یا fingerprint از راه‌حل می‌باشد.

بعد از ارزیابی، نوبت به انجام مراحل انتخاب، ترکیب و جهش می‌باشد. طبق [۱۰]، بهترین روش برای انتخاب راه‌حل‌ها، انتخاب به‌صورت تصادفی می‌باشد که می‌تواند دقت ۹۵.۹۳ درصدی با احتمال خطا ۱۱ صدم درصدی را نتیجه دهد. به این ترتیب، در هر مرحله از انتخاب، دو فرد از جمعیت به‌صورت تصادفی و با احتمال برابر انتخاب می‌شوند. این موضوع می‌تواند سبب افزایش اکتشاف در فضای مسئله شود و ما را از به دام افتادن در یک نقطه بهینه محلی، نجات دهد. با انتخاب دو فرد از جمعیت به صوت تصادفی، نوبت به ترکیب آن‌ها می‌رسد. همان‌طور که قبلاً نیز گفته شد، یک راه‌حل شامل لایه‌های موجود در شبکه به همراه پارامترهای هر لایه می‌باشد. نحوه‌ی ترکیب دو راه‌حل استفاده از تقطیع دوتایی می‌باشد و به این شکل است که ابتدا آن فردی که دارای تعداد لایه‌های کانولوشنی کمتری است، انتخاب می‌شود. سپس، دو نقطه به‌صورت تصادفی انتخاب شده و راه‌حل را به ۳ قسمت تقسیم می‌نماید. در راه‌حل دیگر نیز دو نقطه به‌صورت تصادفی انتخاب می‌شود، با این موضوع که فاصله‌ی بین نقاط انتخاب شده در هردو راه‌حل یکسان می‌باشد. سپس، قسمت‌های ۱ و ۳ از دو راه‌حل با یکدیگر، جابه‌جا شده و دو فرزند جدید ساخته می‌شود. دقت شود که در این فرآیند، تنها لایه‌های کانولوشنی با یکدیگر جابه‌جا شدند. لایه‌های چگال و نرخ یادگیری هر دو راه‌حل نیز با احتمالات متفاوتی نسبت به لایه‌های کانولوشنی با یکدیگر جابه‌جا می‌شوند.

بعد از ساخت دو فرزند جدید، نوبت به جهش آن‌ها می‌رسد. جهش در یک راه‌حل می‌تواند رخ دادن جهش در هر لایه از آن شامل لایه‌های کانولوشنی و چگال صورت گیرد. به‌طورکلی، جهش می‌تواند از بین اعمال اضافه کردن، تغییر و حذف کردن انتخاب شود. هریک از اعمال با یک احتمال مشخص رخ می‌دهند. اعمال زیادی را می‌توان توسط این سه عملگر انجام داد مانند، اضافه نمودن تابع فعال‌سازی، dropout و یا max pooling به یک لایه، اضافه نمودن یک لایه کانولوشنی به شبکه، اضافه نمودن لایه‌های مخفی در شبکه چگال، تغییر در تعداد فیلترهای لایه‌ها، تغییر در تابع فعال‌سازی، تغییر در dropout، حذف نمودن یک لایه کانولوشنی در شبکه، حذف نمودن لایه مخفی در شبکه دسته‌بند، حذف نمودن تابع فعال‌سازی، dropout

و max pooling از یک لایه. هر یک از ترکیبات فوق، با احتمالی مشخص صورت می‌گیرند و طبیعتاً تغییر در احتمالات می‌تواند پاسخ گرفته‌شده در پایان را تحت تأثیر قراردهد.

هر یک از موارد گفته‌شده در الگوریتم‌های شکل‌های (۴)، (۵) و (۶) که برنامه‌ی kNas از آن‌ها استفاده می‌کند، به‌طور کامل مشخص‌شده است.

Algorithm 1 kNas EA Algorithm

Input: Configuration parameters which specify all the necessary parameters such as Population size, Generation number, Crossover probability, All mutation probabilities and etc.

Output: Best neural architecture

1: $P \leftarrow$Initialize the first population
2: Calculate the fitness value of each individual
3: $t \leftarrow 0$
4: **while** $t <$T **do**
5: Two individuals will be selected from the population and will be combined by the algorithm 2 and mutated by the algorithm 3
6: $P_t \leftarrow$P
7: Sort P_t in decreasing order and select the top popSize individuals for the next generation.
8: $t \leftarrow t + 1$
9: **end while**
return First element of the population

شکل ۴: بدنه اصلی الگوریتم

۴ اجرای برنامه‌ی

برنامه kNas از یک فایل تنظیم (Configuration File) جهت مشخص نمودن تمامی متغیرهای مهم در اجرا استفاده می‌کند. این فایل از دو بخش تشکیل‌شده است. یک بخش به مشخص نمودن پارامترهای مربوط به شبکه مانند حداکثر تعداد لایه‌های

Algorithm 2 kNas EA Crossover

Input1: Configuration parameters which specify all the necessary parameters such as Population size, Generation number, Crossover probability, All mutation probabilities and etc.
Input2: ind1
Input3: ind2
Output: New offsprings $off1$ and $off2$

$off1 \leftarrow$ind1
$off2 \leftarrow$ind2
if random() < crossover_probability **then**
    Select two different positions randomly, p1 and p2 in ind1.
    Select two different positions randomly, p3 and p4 in ind2.
    Note that: p1<p2 and p3<p4 and p2-p1 = p4-p3
    $off1$ = ind1[:p1] + ind2[p3:p4] + ind1[p2:]
    $off2$ = ind2[:p3] + ind1[p1:p2] + ind2[p4:]
end if
if random() < dfc_crossover_probability **then**
    Swap($off1$.dfcLayer , $off2$.dfcLayer)
end if
if random() < lr_crossover_probability **then**
    Swap($off1$.learningRate , $off2$.learningRate)
end if
return $off1$,$off2$

شکل ۵: الگوریتم ترکیب

Algorithm 3 kNas EA Mutation

Input1: Configuration parameters which specify all the necessary parameters such as Population size, Generation number, Crossover probability, All mutation probabilities and etc.
Input2: An individual
Output: Mutated individual

for each layer in the individual **do**
    if random() < mutation_probability **then**
        // Mutate the layer based on the probabilities
    end if
end for
return individual

شکل ۶: الگوریتم جهش

کانولوشنی، حداکثر تعداد لایه‌های مخفی در لایه چگال، مقادیر قابل‌قبول برای تعداد فیلترهای لایه کانولوشنی و نورون‌های شبکه چگال و غیره می‌پردازد. بخشی دیگر به مشخص نمودن پارامترهای مربوط به الگوریتم تکاملی، مانند اندازه جمعیت، تعداد نسل‌ها، احتمال ترکیب راه‌حل‌ها، احتمال جهش و احتمال رخ دادن هریک از اعمال جهش معرفی‌شده در بخش قبل می‌پردازد. شکل‌های (۷)، (۸)، (۹) و (۱۰) یک نمونه از این فایل را نشان می‌دهند:

```
# ##################################### #
# Configuration file of the KNAS program  #
#           config v1.0                   #
#          Kiarash Sedghi                 #
# ##################################### #



# ########################### #
#                             #
#   Training Parameters       #
#                             #
# ########################### #

MAX_CNN = 6

BATCH_SIZE = 64

EPOCHS = 2

KERNEL_SIZE = 2

PADDING = 1

STRIDE = 1
```

شکل ۷: فایل تنظیم

```


TRAIN_SPLIT = 0.75
# VAL_SPLIT = 1 - TRAIN_SPLIT

DEVICE = cuda

TRAIN_ROOT_DIR = data

TEST_ROOT_DIR= data

SPLIT_SEED = 42

INPUT_DIM = 32

FILTER_POSS_VALUES = [ 1, 2, 4 , 8, 16, 32, 64, 128 ]

HIDDEN_LAYERS_NEURONS_POSS_VALUE = [ 2, 4 , 8, 16, 32, 64, 128 , 256 , 512 ]

MAX_NUM_HIDDEN_LAYERS = 2
```

شکل ۸: فایل تنظیم (ادامه)

```
51
52  # ##################### #
53  #                       #
54  #   EA Parameters       #
55  #                       #
56  # ##################### #
57
58  POP_SIZE = 2
59
60  GEN_NUM = 1
61
62  CROSS_PROB = 0.3
63
64  CROSS_SWAP_DFC_PROB = 0.5
65
66  MUT_LEARNING_RATE_PROB = 0.4
67
68
69  # Mutation parameters for convolutional layer
70
71  MUT_PROB_CL = 0.9
72
73  MUT_ADD_PROB_CL = 0.3
74  MUT_MOD_PROB_CL = 0.3
75  MUT_REM_PROB_CL = 0.3
76
77  MUT_ADD_CN_LAYER = 0.3
78  MUT_ADD_BATCHNORM_PROB_CL = 0.3
79  MUT_ADD_ACTFUNC_PROB_CL = 0.3
80  MUT_ADD_DROPOUT_PROB_CL = 0.3
81  MUT_ADD_MAXPOOL_PROB_CL = 0.3
82
83  MUT_MOD_ACTFUNC_PROB_CL = 0.3
84  MUT_MOD_DROPOUT_PROB_CL = 0.3
85  MUT_MOD_FILTERS_PROB_CL = 0.3
```

شکل ۹: فایل تنظیم (ادامه)

```
86
87  MUT_REM_CNLAYER_PROB_CL = 0.3
88  MUT_REM_BATCHNORM_PROB_CL = 0.3
89  MUT_REM_ACTFUNC_PROB_CL = 0.3
90  MUT_REM_DROPOUT_PROB_CL = 0.3
91  MUT_REM_MAXPOOL_PROB_CL = 0.3
92
93
94
95  # Mutation parameters for dfc layer
96
97  MUT_PROB_DL = 0.9
98
99
100 MUT_ADD_PROB_DL = 0.3
101 MUT_MOD_PROB_DL = 0.3
102 MUT_REM_PROB_DL = 0.3
103
104 MUT_ADD_HI_LAYER = 0.3
105 MUT_ADD_BATCHNORM_PROB_DL = 0.3
106 MUT_ADD_ACTFUNC_PROB_DL = 0.3
107 MUT_ADD_DROPOUT_PROB_DL = 0.3
108
109
110 MUT_MOD_ACTFUNC_PROB_DL = 0.3
111 MUT_MOD_DROPOUT_PROB_DL = 0.3
112 MUT_MOD_FILTERS_PROB_DL = 0.3
113
114 MUT_REM_HIL_PROB = 0.3
115 MUT_REM_BATCHNORM_PROB_DL = 0.3
116 MUT_REM_ACTFUNC_PROB_DL = 0.3
117 MUT_REM_DROPOUT_PROB_DL = 0.3
118
```

شکل ۱۰: فایل تنظیم (ادامه)

شکل (۱۱) نحوه‌ی اجرای برنامه را نشان می‌دهد.

```
┌─[keagle@parrot]─[~/ai/knas]
└──╼ $./knas -f knas.conf
[INF] CUDA Not Available, Switching To CPU
[INF] Train Root Directory Exists, Loading...
[INF] Test Root Directory Exists, Loading...
[INF] Generating The Initial Population ...
[INF] Training Of The Model Has Been Started
Epoch Iter:  0
```

شکل ۱۱: نمونه اجرای برنامه kNas

شکل‌های (۱۲)، (۱۳)، (۱۴) و (۱۵) خروجی برنامه را در Google Colab برای تکرارهای مختلف نشان می‌دهد.

```
Learning rate:  0.9892939948419691

Fitness:  1.1915503734059565

Test accuracy: 0.1

CN Layers:
----------

Sequential(
  (0): Conv2d(1, 128, kernel_size=(2, 2), stride=(1, 1), padding=(1, 1))
  (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (2): Sigmoid()
  (3): Dropout(p=0.8150310139440442, inplace=False)
  (4): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1, ceil_mode=False)
)
Sequential(
  (0): Conv2d(128, 128, kernel_size=(2, 2), stride=(1, 1), padding=(1, 1))
  (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (2): ReLU()
  (3): Dropout(p=0.20889489326510435, inplace=False)
)
Sequential(
  (0): Conv2d(128, 128, kernel_size=(2, 2), stride=(1, 1), padding=(1, 1))
  (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (2): Sigmoid()
  (3): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1, ceil_mode=False)
)
Sequential(
  (0): Conv2d(128, 4, kernel_size=(2, 2), stride=(1, 1), padding=(1, 1))
  (1): BatchNorm2d(4, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (2): Sigmoid()
  (3): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1, ceil_mode=False)
)
Sequential(
  (0): Conv2d(4, 8, kernel_size=(2, 2), stride=(1, 1), padding=(1, 1))
  (1): ReLU()
  (2): Dropout(p=0.6349495325671968, inplace=False)
  (3): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1, ceil_mode=False)
)
DFC Layer:
Sequential(
  (0): Linear(in_features=72, out_features=10, bias=True)
)
```

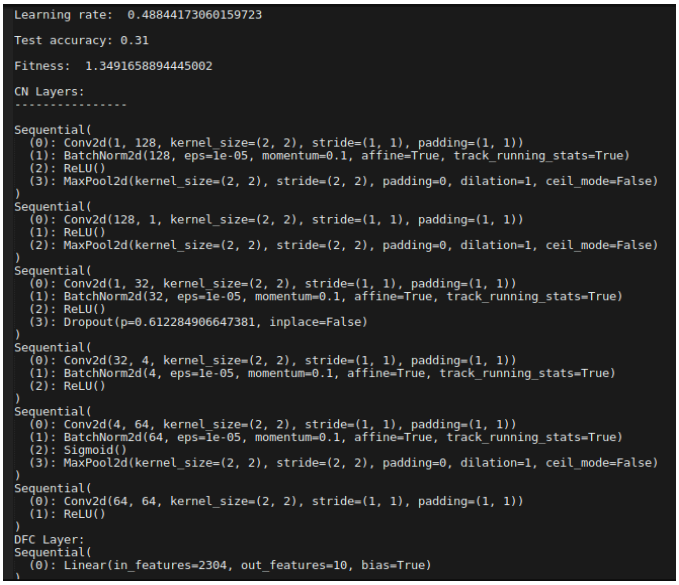
شکل ۱۲: نمونه اجرای اول

```
Learning rate:  0.6382773108747426

Test accuracy: 0.09


Fitness:  1.235749196047905
CN Layers:
---------------

Sequential(
  (0): Conv2d(1, 128, kernel_size=(2, 2), stride=(1, 1), padding=(1, 1))
  (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (2): Dropout(p=0.7052843485536228, inplace=False)
  (3): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1, ceil_mode=False)
)
Sequential(
  (0): Conv2d(128, 1, kernel_size=(2, 2), stride=(1, 1), padding=(1, 1))
  (1): Dropout(p=0.18463227398602888, inplace=False)
  (2): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1, ceil_mode=False)
)
DFC Layer:
Sequential(
  (0): Linear(in_features=64, out_features=32, bias=True)
  (1): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (2): Sigmoid()
  (3): Dropout(p=0.45662457259260425, inplace=False)
  (4): Linear(in_features=32, out_features=10, bias=True)
)
```

شکل ۱۳: نمونه اجرای دوم

```
Learning rate:  0.8122398818064219

Test accuracy: 0.54



Fitness:  1.669785499522742
CN Layers:
---------------

Sequential(
  (0): Conv2d(1, 32, kernel_size=(2, 2), stride=(1, 1), padding=(1, 1))
  (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (2): Sigmoid()
  (3): Dropout(p=0.5237773279191128, inplace=False)
  (4): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1, ceil_mode=False)
)
Sequential(
  (0): Conv2d(32, 32, kernel_size=(2, 2), stride=(1, 1), padding=(1, 1))
  (1): ReLU()
  (2): Dropout(p=0.24713718836195375, inplace=False)
)
DFC Layer:
Sequential(
  (0): Linear(in_features=9248, out_features=10, bias=True)
)
```

شکل ۱۴: نمونه اجرای سوم



شکل ۱۵: نمونه اجرای چهارم

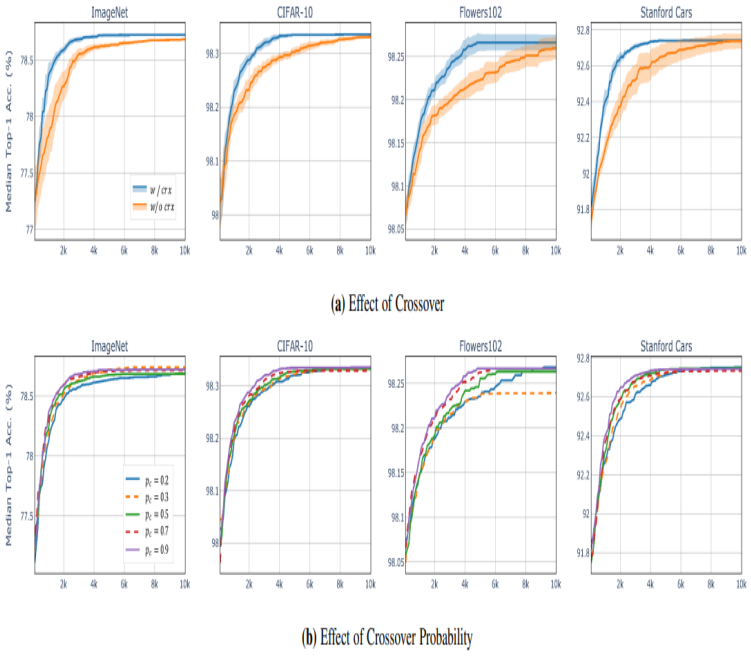
در اجراهای فوق، پارامترهای اندازه جمعیت مقدار ۵، تعداد نسل مقدار ۵ و پارامتر Epoch مقدار ۲ را اختیار کرده بودند. مقدار دقیق هر یک از پارامترها در فایل knas.conf که یک فایل نمونه تنظیم است و در آدرس گیتهاب پروژه [۱۲] قرار دارد، قابل دسترسی است.

۵ تحلیل

به دلیل کمبود وقت، تأثیر تمامی پارامترهای مشخص‌شده در دستیابی به شبکه عصبی که دارای بیشترین دقت است، در این گزارش بررسی نشده است. در این گزارش، تأثیر کلی احتمال جهش و ترکیب به‌صورت کلی بررسی‌شده است.

عملگر ترکیب یکی از مهم‌ترین عملگرهای الگوریتم‌های تکاملی می‌باشد. در اولین الگوریتم‌های ارائه‌شده برای NAS که مبتنی بر الگوریتم‌های تکاملی بودند، عملگر ترکیب به‌طورکلی حذف شده بود. اما با انتخاب مقادیر مناسب برای آن، می‌توان تأثیر مثبت آن را در الگوریتم مشاهده نمود. شکل (۱۶) به‌خوبی تأثیر وجود عملگر ترکیب را نشان می‌دهد. همان‌طور که مشخص است، با ارزیابی الگوریتم بر روی دیتاست‌های متفاوت، به این نتیجه می‌توان رسید که وجود عملگر ترکیب می‌تواند

دقت را افزایش دهد. همچنین در قسمت دوم شکل (۱۶)، مشخص است که افزایش احتمال ترکیب، به‌طور میانگین بر روی دیتاست‌های متفاوت، سبب افزایش دقت شده است.

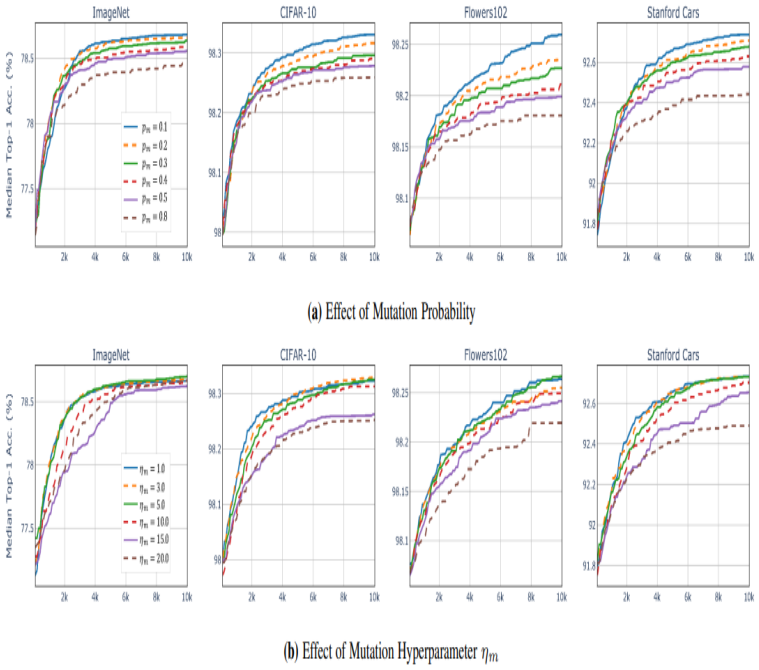


شکل ۱۶: تاثیر عملگر ترکیب [۱۱]

شکل (۱۷) نیز تأثیر جهش را نشان داده است. همان‌طورکه در قسمت اول شکل مشخص است، افزایش احتمال جهش، تأثیر منفی بر دقت مدل در ارزیابی با دیتاست‌های مختلف داشته است.

۶ منبع برنامه kNas

برای دسترسی به منبع کد برنامه kNas به [۱۲] مراجعه شود.



شکل ۱۷: تاثیر عملگر جهش [۱۱]

منابع

[1] Ren, Pengzhen, et al. "A comprehensive survey of neural architecture search: Challenges and solutions." arXiv preprint arXiv:2006.02903 (2020).

[2] Xue, Yu & Jiang, Pengcheng & Neri, Ferrante & Liang, Jiayu. (2021). A Multi-Objective Evolutionary Approach Based on Graph-in-Graph for Neural Architecture Search of Convolutional Neural Networks. International Journal of Neural Systems. 31. 2150035. 10.1142/S0129065721500350.

[3] Luo, Renqian, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. "Neural architecture optimization." arXiv preprint arXiv:1808.07233 (2018).

[4] Scott E Fahlman and Christian Lebiere. The cascade-correlation learning architecture. In Advances in neural information processing systems, pages 524–532, 1990.

[5] Hiroaki Kitano. Designing neural networks using genetic algorithms with graph generation system. Complex Systems Journal, 4:461–476, 1990.

[6] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. arXiv preprint arXiv:1611.01578, 2016.

[7] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recogniti

[8] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. arXiv preprint arXiv:1802.01548, 2018.

[9] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In International Conference on Machine Learning, pages 2902–2911, 2017.

[10] Xu, Y., Wang, Y., Han, K., Tang, Y., Jui, S., Xu, C., & Xu, C. (2021). ReNAS: Relativistic evaluation of neural architecture search. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 4411-4420).

[11] Lu, Zhichao, Gautam Sreekumar, Erik Goodman, Wolfgang Banzhaf, Kalyanmoy Deb, and Vishnu Naresh Boddeti. "Neural architecture transfer." IEEE Transactions on Pattern Analysis and Machine Intelligence (2021).

[12] https://github.com/keagleV/kNas