چالش سوم: Abstraction and Reasoning Challenge

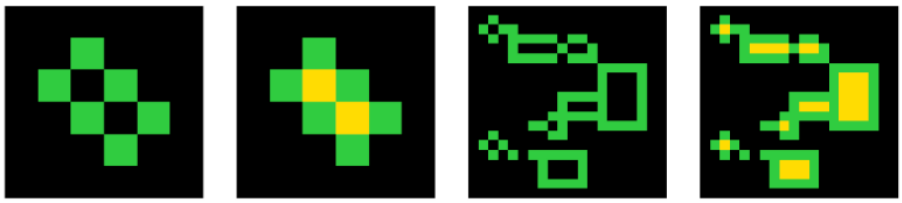
فراهم‌آورنده: سینا حیدری

زمستان ۹۹

برای مشاهده‌ی جزئیات این چالش در کگل به لینک زیر مراجعه کنید.

https://www.kaggle.com/c/abstraction-and-reasoning-challenge/overview

۱ در مورد این چالش



آیا یک کامپیوتر می‌تواند وظایف پیچیده و انتزاعی را از چند نمونه مشخص یاد بگیرد؟
تکنیک‌های **یادگیری ماشین** فعلی شکننده و گرسنه داده هستند-فقط قادر به شناسایی الگوهایی است که قبلا دیده است. با استفاده از روش‌های کنونی، یک الگوریتم با دسترسی به حجم داده بالا می‌تواند مهارت جدید کسب کند؛ اما توانایی‌های شناختی که بتوان آن‌ها را به وظایف زیادی تعمیم داد هنوز کار دشواری است. همین موجب چالش در ساختن سیستم‌هایی که بتوانند از پس تغییر و غیر قابل پیش‌بینی بودن دنیای واقعی بر بیایند شده است. نمونه‌هایی از این تکنولوژی‌ها روبات‌ها و خودروهای خودران می‌باشد.
در هر صورت، رویکردهای دیگری همچون: inductive programming بالقوه‌یی برای برداشت و استدلال انسان‌گونه فراهم می‌آورد. ARC[1] فراهم‌آورنده یک معیار اندازه‌گیری مهارت‌آموزی برای وظایف نامشخص است. محدودیت این این مجموعه‌داده، کم بودن نتایجی است که نشان‌دهنده یادگیری وظایف پیچیده می‌باشند. می‌توان ARC را به عنوان یک نگاه اجمالی به آینده که در آن هوش‌مصنوعی می‌تواند به سرعت آموزش دیده و مشکلات مربوط به خودش را حل کند. این چالش از ما دعوت کرده تا در این آینده سهیم بوده و آینده را به حال بیاوریم. این چالش توسط **François Chollet**-پدیدآورنده شبکه‌های عصبی Keras-میزبانی شده است. تالیف Chollet در مورد اندازه‌گیری هوش، نگاهی به موضوعیت و انگیزه معیار ARC می‌اندازد.
در این مسابقه شما هوش‌مصنوعی‌یی خواهید ساخت که توانایی انجام وظایفی کاملا جدید را داشته باشد. هر وظیفه ARC شامل سه تا پنج جفت نمونه ورودی و خروجی آموزشی، و یک ورودی آزمایشی است. برای ورودی آزمایشی باید خروجی مربوطه را با بهره‌گیری از الگوی یادگرفته شده از نمونه آموزشی پیش‌بینی کنید.
اگر موفق شوید، به نزدیک کردن کامپیوترها را به ادراک/شناخت انسان کمک خواهید کرد و در مسیر کاربردهای کاملا جدید هوش‌مصوعی قرار دارید.
برای مشاهده مقاله آقای **François Chollet** لینک زیر مراجعه کنید:

https://arxiv.org/abs/1911.01547

[1] The Abstraction and Reasoning Corpus

۲ در مورد این گزارش

به چالش ARC[۲]-یک بالقوه برای دستیابی به AGI[۳]-خوش آمدید. در این رقابت، ما به چالش کشیده شده‌ایم تا یک الگوریتم طراحی کنیم که قادر به انجام وظایف استدلال کردنی‌یی باشد که قبلا ندیده. مسائل کلاسیک یادگیری ماشین عموما یک وظیفه مشخص را شامل می‌شوند که با انجام یادگیری از میلیون‌ها نمونه داده قابل حل است. اما در این رقابت، ما یک الگوریتم که می‌تواند الگوها را از روی تعداد کمی نمونه یاد بگیرد، طراحی می‌کنیم. در این گزارش نشان خواهیم داد: چگونه می‌توان از data augmentation و supervised machine learning برای ساختن یک مدل پایه، برای حل این مسئله استفاده کرد.

۳ بررسی اکتشافی داده

بارگذاری کتابخانه‌های مورد نیاز

```
import os
import gc
import cv2
import json
import time

import numpy as np
import pandas as pd
from pathlib import Path
from keras.utils import to_categorical

import seaborn as sns
import plotly.express as px
from matplotlib import colors
import matplotlib.pyplot as plt
import plotly.figure_factory as ff

import torch
T = torch.Tensor
import torch.nn as nn
from torch.optim import Adam
from torch.utils.data import Dataset, DataLoader
```

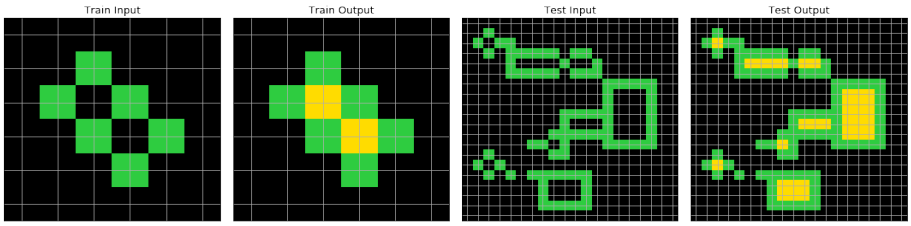
بیایید به نمونه‌های زوج ورودی-خروجی در داده‌های آموزشی و آزمایشی را نگاه کنیم:

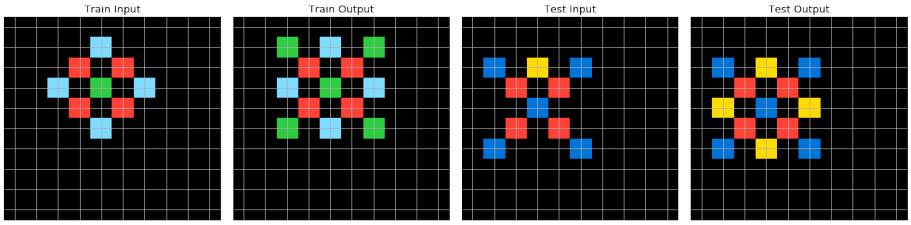
```
data_path = Path('/kaggle/input/abstraction-and-reasoning-challenge/')
training_path = data_path / 'training'
```

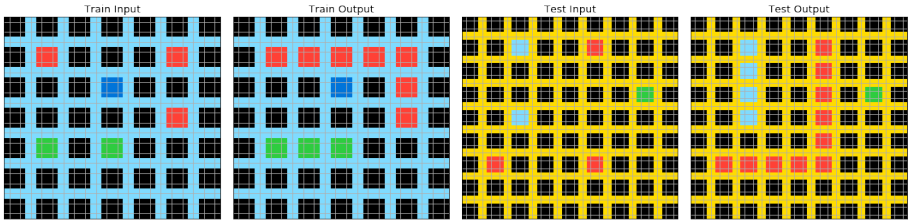
[۲] Abstraction and Reasoning Challenge
[۳] Artificial General Intelligence

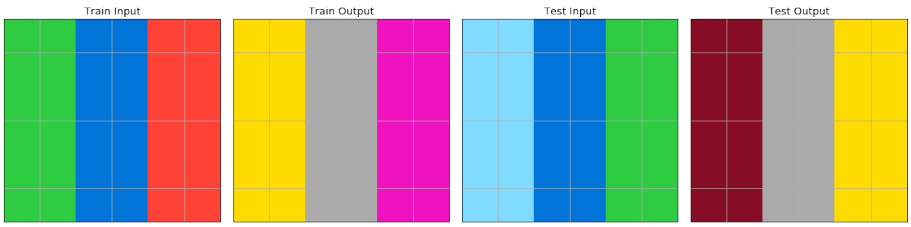
```
training_tasks = sorted(os.listdir(training_path))

for i in [1, 19, 8, 15, 9]:

  task_file = str(training_path / training_tasks[i])

  with open(task_file, 'r') as f:
  task = json.load(f)

  def plot_task(task):
    """
    Plots the first train and test pairs of a specified task,
    using same color scheme as the ARC app
    """
    cmap = colors.ListedColormap(
    ['#000000', '#0074D9','#FF4136','#2ECC40','#FFDC00',
    '#AAAAAA', '#F012BE', '#FF851B', '#7FDBFF', '#870C25'])
    norm = colors.Normalize(vmin=0, vmax=9)
    fig, ax = plt.subplots(1, 4, figsize=(15,15))
    ax[0].imshow(task['train'][0]['input'], cmap=cmap, norm=norm)
    width = np.shape(task['train'][0]['input'])[1]
    height = np.shape(task['train'][0]['input'])[0]
    ax[0].set_xticks(np.arange(0,width))
    ax[0].set_yticks(np.arange(0,height))
    ax[0].set_xticklabels([])
    ax[0].set_yticklabels([])
    ax[0].tick_params(length=0)
    ax[0].grid(True)
    ax[0].set_title('Train Input')
    ax[1].imshow(task['train'][0]['output'], cmap=cmap, norm=norm)
    width = np.shape(task['train'][0]['output'])[1]
    height = np.shape(task['train'][0]['output'])[0]
    ax[1].set_xticks(np.arange(0,width))
    ax[1].set_yticks(np.arange(0,height))
    ax[1].set_xticklabels([])
    ax[1].set_yticklabels([])
    ax[1].tick_params(length=0)
    ax[1].grid(True)
    ax[1].set_title('Train Output')
    ax[2].imshow(task['test'][0]['input'], cmap=cmap, norm=norm)
    width = np.shape(task['test'][0]['input'])[1]
    height = np.shape(task['test'][0]['input'])[0]
    ax[2].set_xticks(np.arange(0,width))
    ax[2].set_yticks(np.arange(0,height))
    ax[2].set_xticklabels([])
    ax[2].set_yticklabels([])
    ax[2].tick_params(length=0)
    ax[2].grid(True)
    ax[2].set_title('Test Input')
    ax[3].imshow(task['test'][0]['output'], cmap=cmap, norm=norm)
    width = np.shape(task['test'][0]['output'])[1]
    height = np.shape(task['test'][0]['output'])[0]
    ax[3].set_xticks(np.arange(0,width))
    ax[3].set_yticks(np.arange(0,height))
    ax[3].set_xticklabels([])
    ax[3].set_yticklabels([])
    ax[3].tick_params(length=0)
```

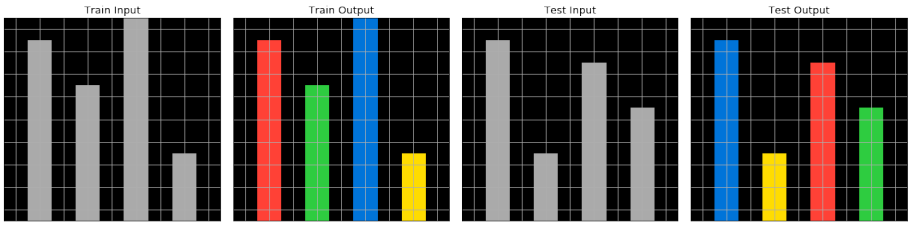
```
    ax[3].grid(True)
    ax[3].set_title('Test Output')
    plt.tight_layout()
    plt.show()

plot_task(task)
```





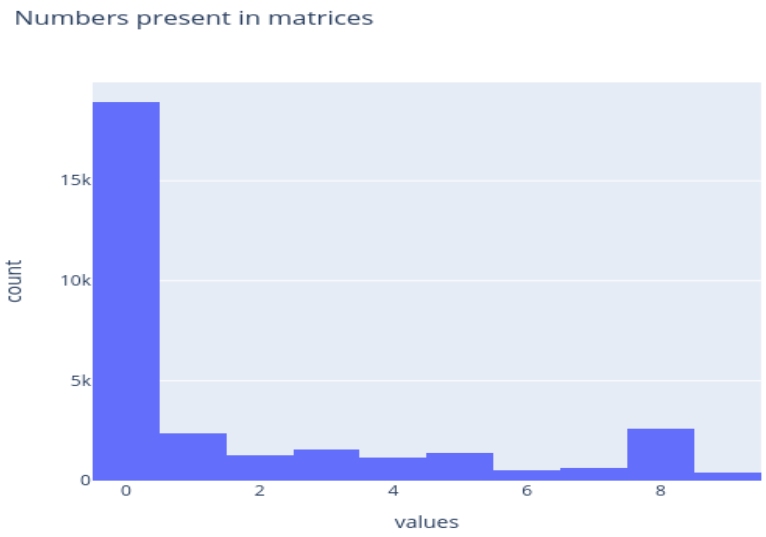






Number frequency

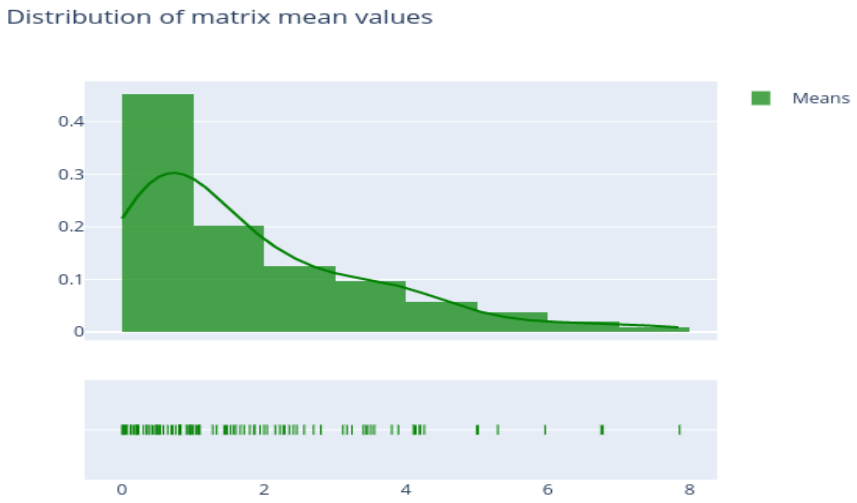
```
1 px.histogram(df, x="values", title="Numbers present in matrices")
```



در گراف بالا به وضوح می‌توان دید که توزیع اعداد دارای چولگی مثبت است. بسیاری از اعداد در ماتریس‌ها صفر می‌باشد و دلیل اصلی غلبه رنگ سیاه در بیشتر ماتریس‌ها است.

Matrix mean values

```
means = [np.mean(X) for X in matrices]
fig = ff.create_distplot([means], group_labels=["Means"], colors=["
    green"])
fig.update_layout(title_text="Distribution of matrix mean values")
```



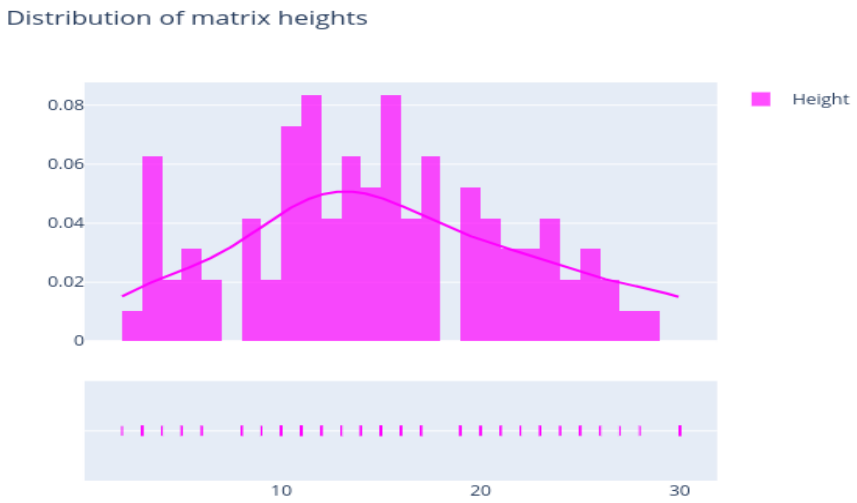
در گراف بالا مشاهده می‌کنید، میانگین‌های پایین متداول‌تر از میانگین‌های بالاست. این گراف هم دارای چولگی مثبت می‌باشد و باز دلیل آن غلبه رنگ سیاه در ماتریس‌ها است.

Matrix heights and width

```
# List comprehension for evaluating matix heights
heights = [np.shape(matrix)[0] for matrix in matrices]
# List comprehension for evaluating matix width
widths = [np.shape(matrix)[1] for matrix in matrices]
```

Matrix heights

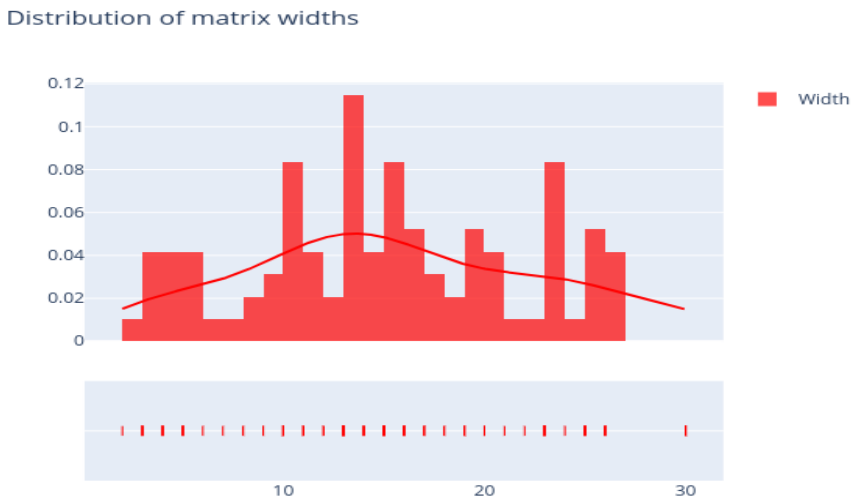
```
fig = ff.create_distplot([heights], group_labels=["Height"], colors=["
    magenta"])
fig.update_layout(title_text="Distribution of matrix heights")
```



از گراف بالا نتیجه می‌گیریم که ارتفاع ماتریس‌ها توزیع به مراتب یکنواخت‌تری دارند (با چولگی بسیار کمتر). این توزیع نرمال است و دارای میانگین تقریبا ۱۵ می‌باشد.

Matrix width

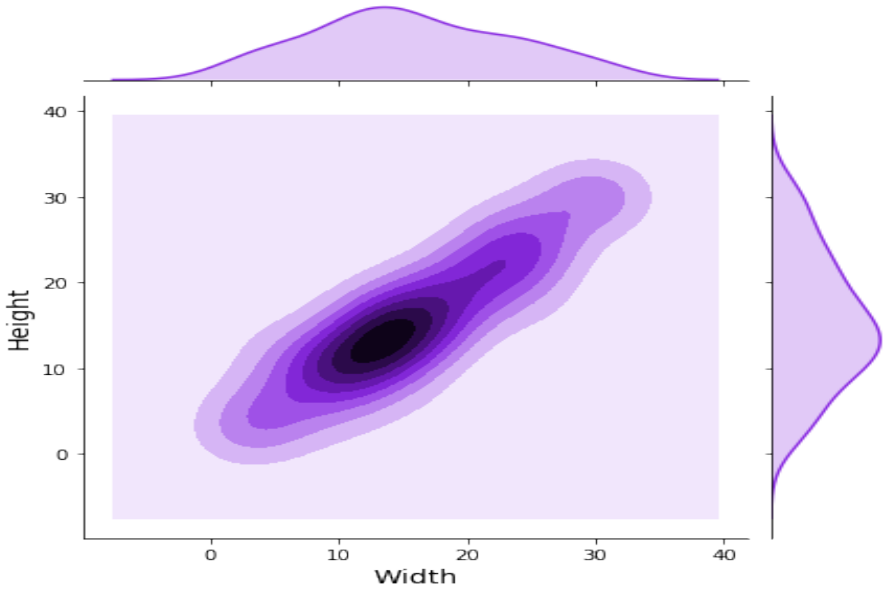
```
fig = ff.create_distplot([widths], group_labels=["Width"], colors=["red
    "])
fig.update_layout(title_text="Distribution of matrix widths")
```



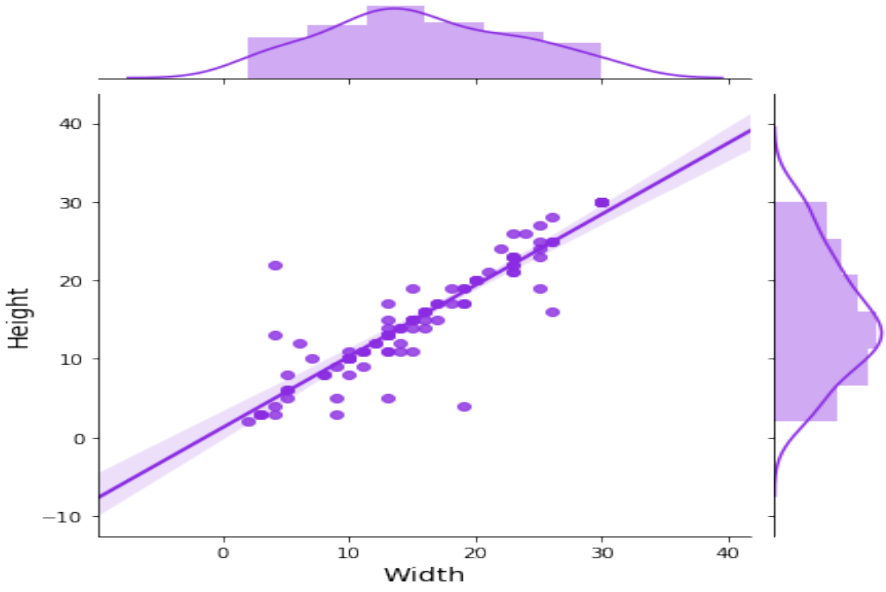
در گراف بالا طول ماتریس‌ها یکنواخت بوده و میانگین آن تقریبا ۱۶ می‌باشد.

Height vs. Width

```
plot = sns.jointplot(widths, heights, kind="kde", color="blueviolet")
plot.set_axis_labels("Width", "Height", fontsize=14)
plt.show(plot)
```




```
plot = sns.jointplot(widths, heights, kind="reg", color="blueviolet")
plot.set_axis_labels("Width", "Height", fontsize=14)
plt.show(plot)
```



از گراف‌های بالا نتیجه می‌گیریم که طول و ارتفاع ماتریس‌ها هم‌بستگی مثبت دارند، این یعنی: ماتریس با ارتفاع بیشتر عموما طول بیشتری هم دارد. دلیل این امر مربعی بودن بیشتر ماتریس‌ها است.

۴ راه حل

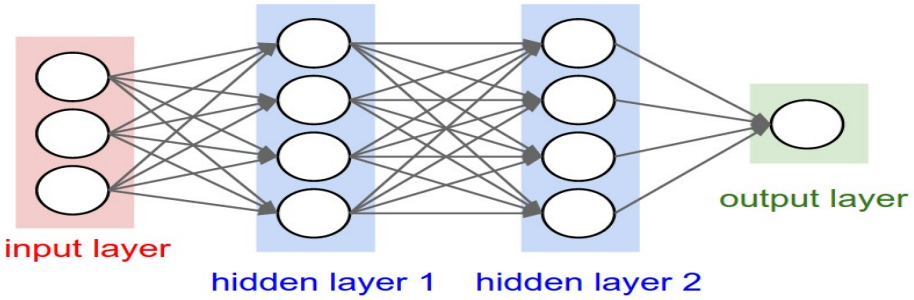
رویکرد ما از تکنیک‌های data augmentation و یک مدل دو بعدی باناظر CNN بهره می‌گیرد. همانطور که می‌دانیم، تصویرها صرفا آرایه‌های سه بعدی از مقدار روشنایی هستند. با استفاده از CNN می‌خواهیم مدل‌مان، ویژگی ها و سلسه‌مراتب ویژگی‌ها را مستقیما از نمونه‌ها یاد بگیرد. مراحلی که در طراحی و کاربرد مدل‌های CNN–و به طور کلی بینایی ماشین–وجود دارد را در زیر می‌بینید:

۱. Domain knowledge

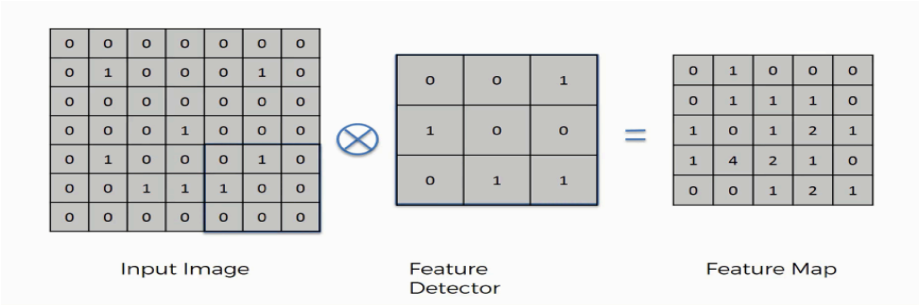
۲. Define features

۳. Detect features to classify

اگر به طور ساده به مسئله نگاه کنیم، می‌توان از شبکه عصبی fully connected برای تشخیص ویژگی‌های دیداری استفاده کرد. این شبکه عصبی، تصویرهای دوبعدی یا بردار مقادیر پیکسل‌ها را به عنوان ورودی شبکه عصبی دریافت می‌کند. هر نورون در لایه پنهان به تمام نورون‌های لایه ورودی متصل می‌شود. دو مشکل اصلی برای این نوع از شبکه‌های عصبی، نداشتن اطلاعات فضایی (spatial) و زیاد بودن پارامترها می‌باشد.



ما می‌خواهیم ساختار فضایی را برای این تصویرها تشخیص دهیم؛ برای این کار تکه‌هایی (patches) از ورودی را به نورون‌های لایه پنهان متصل می‌کنیم. هر تکه از ورودی به یک نورون واحد در لایه بعدی متصل شده و با یک پنجره لغزان ارتباط‌ها را تعریف می کنیم.



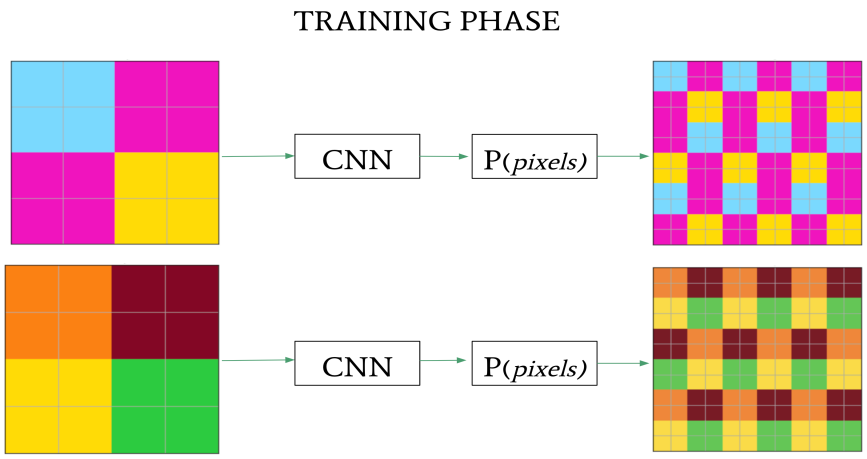
در تصویر بالا، feature detector همان فیلتر ما می‌باشد که هر عنصر آن به‌صورت پیکسل در پیکسل با عنصرهای پنجره لغزان ضرب خواهد شد (elementwise multiplication). به این عمل convolution operation گفته می‌شود. در نهایت می‌توانیم عملکرد شبکه عصبی را به سه مرحله زیر خلاصه کنیم:

۱. اعمال مجموعه‌یی از وزن‌ها-یا همان فیلتر-برای استخراج ویژگی‌های محلی

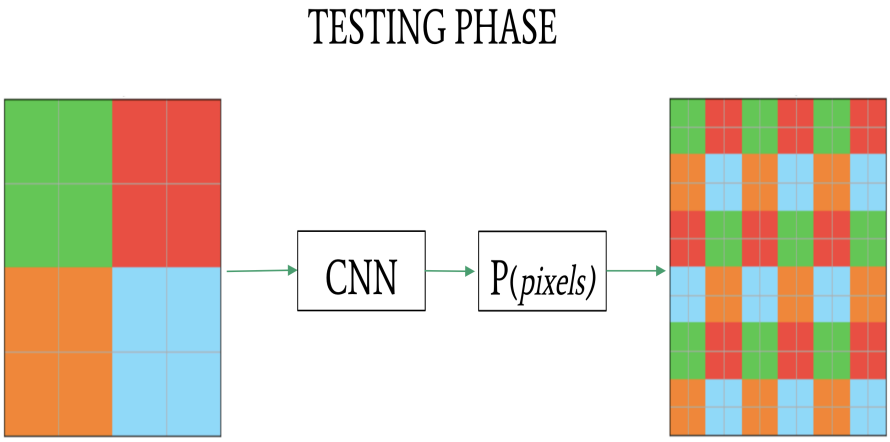
۲. استفاده از فیلترهای متفاوت برای استخراج ویژگی‌های متفاوت

۳. اشتراک‌گذاری پارامترهای هر فیلتر به‌صورت فضایی

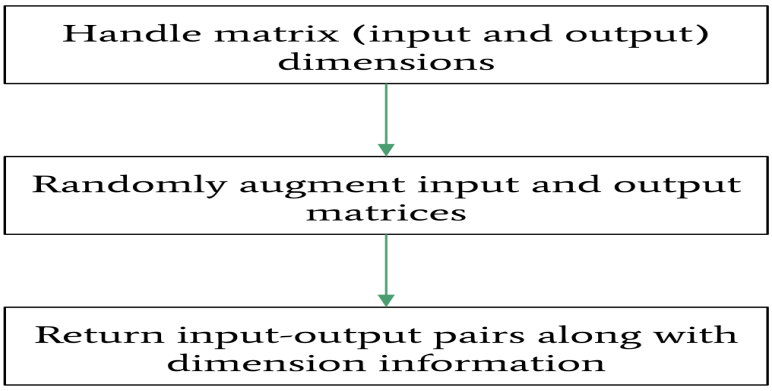
مدل ما، یک ماتریس دوبعدی را به‌عنوان ورودی دریافت کرده و احتمال softmax برای مقادیر گوناگون را به دست می‌آورد. به دلیل اینکه تعداد محدودی نمونه آموزشی برای هر وظیفه وجود دارد، زوج‌های ورودی-خروجی جدیدی را با تغییر تصادفی رنگ‌ها تولید می‌کنیم.



در نمودار بالا، زوج‌های آزمایشی تقویت شده‌اند تا یک مجموعه‌داده بزرگ‌تر تولید شود. از این مجموعه‌داده برای آموزش مدل CNN برای هر وظیفه بهره می‌بریم. CNN توزیع احتمال بر روی پیکسل ها یا مقادیر ماتریس را پیش‌بینی می‌کند. از این توزیع احتمال هم برای تولید خروجی ماتریس نهایی استفاده می‌شود.



پردازش داده



مراحل اولیه در خط لوله پردازش داده را در تصویر بالا مشاهده می‌کنید. این مراحل را می‌توان به گونه زیر خلاصه‌سازی کرد.

۱. **Handle matrix (input and output) dimensions**
اطمینان حاصل کردن برای یکنواخت بودن ابعاد در ورودی‌ها و خروجی‌ها

۲. **Randomly augment input and output matrices**
جهش دادن (mutation) مقادیر ماتریس برای تولید داده جدید برای هر وظیفه

۳. **Return input-output pairs along with dimension information**
برگرداندن داده X-y به همراه اطلاعات ابعاد

Helper functions

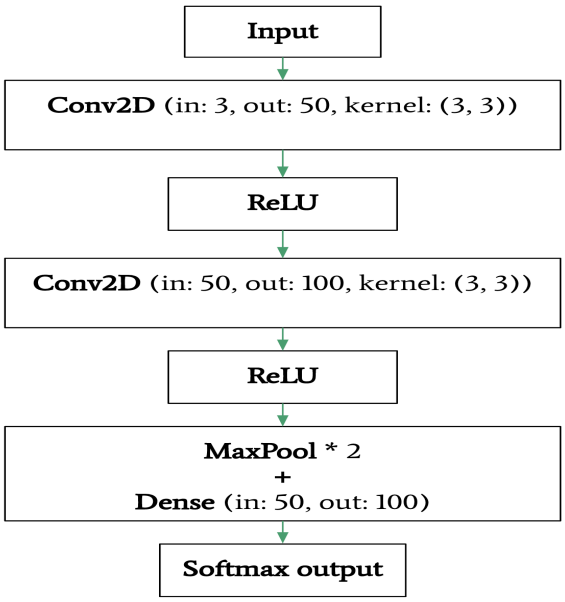
```
def replace_values(a, d):
  return np.array([d.get(i, -1) for i in range(a.min(), a.max() + 1)])[
    a - a.min()]

def repeat_matrix(a):
  return np.concatenate([a]*((SIZE // len(a)) + 1))[:SIZE]

def get_new_matrix(X):
  if len(set([np.array(x).shape for x in X])) > 1:
    X = np.array([X[0]])
  return X

def get_outp(outp, dictionary=None, replace=True):
  if replace:
    outp = replace_values(outp, dictionary)

  outp_matrix_dims = outp.shape
  outp_probs_len = outp.shape[0]*outp.shape[1]*10
  outp = to_categorical(outp.flatten(), num_classes=10).flatten()

  return outp, outp_probs_len, outp_matrix_dims
```

PyTorch DataLoader

```
class ARCDataset(Dataset):
  def __init__(self, X, y, stage="train"):
    self.X = get_new_matrix(X)
    self.X = repeat_matrix(self.X)

    self.stage = stage
    if self.stage == "train":
      self.y = get_new_matrix(y)
      self.y = repeat_matrix(self.y)

  def __len__(self):
    return SIZE

  def __getitem__(self, idx):
    inp = self.X[idx]
    if self.stage == "train":
      outp = self.y[idx]

    if idx != 0:
      rep = np.arange(10)
      orig = np.arange(10)
      np.random.shuffle(rep)
      dictionary = dict(zip(orig, rep))
      inp = replace_values(inp, dictionary)
      if self.stage == "train":
        outp, outp_probs_len, outp_matrix_dims = get_outp(outp,
    dictionary)

    if idx == 0:
```

```
      if self.stage == "train":
        outp, outp_probs_len, outp_matrix_dims = get_outp(outp, None,
    False)

    return inp, outp, outp_probs_len, outp_matrix_dims, self.y
```

مدل‌سازی



ما از یک مدل CNN که ورودی دوبعدی دریافت کرده و خروجی دو بعدی نتیجه می‌دهد، بهره می‌بریم. معماری ترتیبی طبق موارد زیر می‌باشد:

1. **(Conv2D + ReLU) x 2**
2. **MaxPool x 2**
3. **Dense**
4. **Softmax**

احتمال‌های softmax از طریق توابع argmax و resize به ماتریس نهایی تبدیل می‌شود.

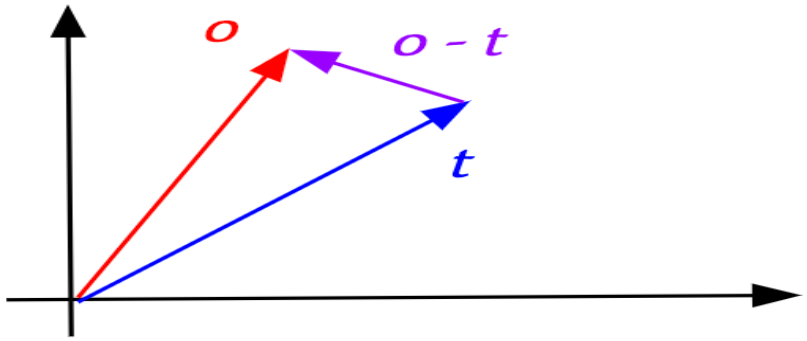
PyTorch CNN model

```
class BasicCNNModel(nn.Module):
  def __init__(self, inp_dim=(10, 10), outp_dim=(10, 10)):
    super(BasicCNNModel, self).__init__()
```

```
    CONV_IN = 3
    KERNEL_SIZE = 3
    DENSE_IN = CONV_OUT_2
    self.relu = nn.ReLU()
    self.softmax = nn.Softmax(dim=1)
    self.dense_1 = nn.Linear(DENSE_IN, outp_dim[0]*outp_dim[1]*10)

    if inp_dim[0] < 5 or inp_dim[1] < 5:
    KERNEL_SIZE = 1

    self.conv2d_1 = nn.Conv2d(CONV_IN, CONV_OUT_1, kernel_size=
    KERNEL_SIZE)
    self.conv2d_2 = nn.Conv2d(CONV_OUT_1, CONV_OUT_2, kernel_size=
    KERNEL_SIZE)

    def forward(self, x, outp_dim):
    x = torch.cat([x.unsqueeze(0)]*3)
    x = x.permute((1, 0, 2, 3)).float()
    self.conv2d_1.in_features = x.shape[1]
    conv_1_out = self.relu(self.conv2d_1(x))
    self.conv2d_2.in_features = conv_1_out.shape[1]
    conv_2_out = self.relu(self.conv2d_2(conv_1_out))

    self.dense_1.out_features = outp_dim
    feature_vector, _ = torch.max(conv_2_out, 2)
    feature_vector, _ = torch.max(feature_vector, 2)
    logit_outputs = self.dense_1(feature_vector)

    out = []
    for idx in range(logit_outputs.shape[1]//10):
    out.append(self.softmax(logit_outputs[:, idx*10: (idx+1)*10]))
    return torch.cat(out, axis=1)
```

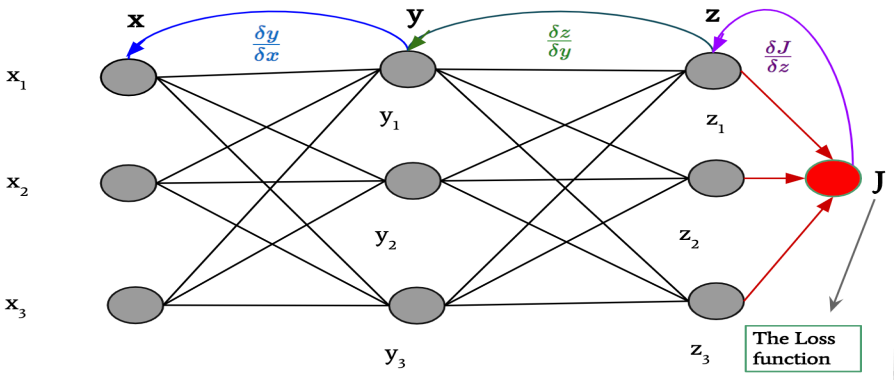
ما مدل را با استفاده از عملکرد autograd در PyTorch آموزش خواهیم داد. به طور مشخص، ما از بهینه‌ساز Adam و تابع هدررفتگی MSE بهره می‌بریم.

Loss (MSE)



همانطور که در بالا نشان داده شده، بردار هدف t و بردار خروجی o با هم اختلاف دارند. تابع هدررفتگی، زاویه‌یی که این دو بردار با هم دارند را محاسبه می‌کند. در اینجا t احتمال واقعی پیکسل و o احتمال پیش‌بینی شده پیکسل است. در داخل کد خط: train_loss = nn.MSELoss()(train_preds, train_y) هدررفتگی MSE را محاسبه می‌کند.

Backpropagation and optimization (Adam)



در نمودار بالا مشاهده می‌شود که از قانون Newton's Chain برای محاسبه gradient برای تابع هدررفتگی استفاده شده است.

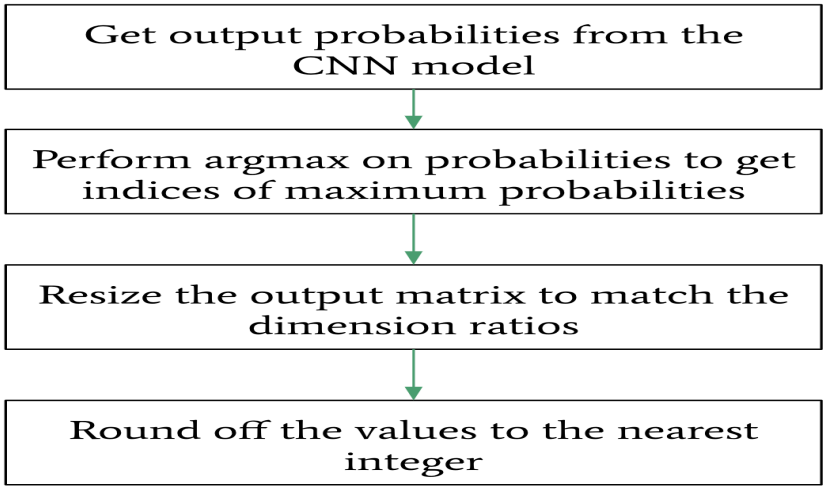
$$\text{Fastest descent direction} = -\nabla J(W,b) = \begin{bmatrix} -\frac{\delta J}{\delta W} \\ -\frac{\delta J}{\delta b} \end{bmatrix}$$

$$\text{Therefore:} \quad \boxed{\begin{aligned} W &:= W - \alpha \cdot \frac{\delta J}{\delta W} \\ b &:= b - \alpha \cdot \frac{\delta J}{\delta b} \end{aligned}}$$

از تابع عمومی به‌روزرسانی در بالا، برای بهینه‌سازی پارامترها با gradient-های محاسبه شده، بهره می‌بریم. به خاطر داشته باشید که الگوریتم‌های پیچیده‌تر مثل: Adam از توابع به‌روزرسانی پیچیده‌تری نسبت به تابع بالا استفاده می‌کنند. در داخل کد، خط train_loss.backward() و optimizer.step() به ترتیب، عمل backpropagation و بهینه‌سازی را انجام می‌دهد.

توابع یاری‌گر

```
def transform_dim(inp_dim, outp_dim, test_dim):
  return (test_dim[0]*outp_dim[0]/inp_dim[0],
    test_dim[1]*outp_dim[1]/inp_dim[1])

def resize(x, test_dim, inp_dim):
  if inp_dim == test_dim:
    return x
  else:
    return cv2.resize(flt(x), inp_dim, interpolation=cv2.INTER_AREA)

def flt(x): return np.float32(x)
def npy(x): return x.cpu().detach().numpy()
def itg(x): return np.int32(np.round(x))
```



1. **Get output probabilites from the CNN model:**
   npy(network.forward(T(X).unsqueeze(0), out_d))
2. **Perform argmax on probabilities to get indices of maximum prbabilities:**
   np.argmax(test_preds.reshape((10, *outp_dim)), axis=0)
3. **Resize the output matrix to match the dimension ratios and round off**
   itg(resize(test_preds, np.shape(test_preds), tuple(itg(transform_dim(inp_dim, outp_dim, test_dim)))))

```
idx = 0
start = time.time()
test_predictions = []

for X_train, y_train in zip(Xs_train, ys_train):
  print("TASK " + str(idx + 1))

  train_set = ARCDataset(X_train, y_train, stage="train")
  train_loader = DataLoader(train_set, batch_size=BATCH_SIZE, shuffle=
    True)

  inp_dim = np.array(X_train[0]).shape
  outp_dim = np.array(y_train[0]).shape
  network = BasicCNNModel(inp_dim, outp_dim).cuda()
  optimizer = Adam(network.parameters(), lr=0.01)

  for epoch in range(EPOCHS):
```

```
    for train_batch in train_loader:
      train_X, train_y, out_d, d, out = train_batch
      train_preds = network.forward(train_X.cuda(), out_d.cuda())
      train_loss = nn.MSELoss()(train_preds, train_y.cuda())

      optimizer.zero_grad()
      train_loss.backward()
      optimizer.step()

  end = time.time()
  print("Train loss: " + str(np.round(train_loss.item(), 3)) + "    " +\
"Total time: " + str(np.round(end - start, 1)) + " s" + "\n")

  X_test = np.array([resize(flt(X), np.shape(X), inp_dim) for X in
    Xs_test[idx-1]])
  for X in X_test:
    test_dim = np.array(T(X)).shape
    test_preds = npy(network.forward(T(X).unsqueeze(0).cuda(), out_d.
    cuda()))
    test_preds = np.argmax(test_preds.reshape((10, *outp_dim)), axis=0)
    test_predictions.append(itg(resize(test_preds, np.shape(test_preds)
    ,
    tuple(itg(transform_dim(inp_dim,
    outp_dim,
    test_dim))))))
    idx += 1
```