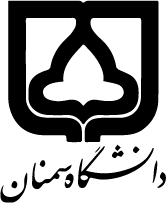
****

****

**دانشکده مهندسی برق و کامپیوتر - گروه مهندسی کامپیوتر**

**پایان نامه پروژه کارشناسی**

**رشته : مهندسی کامپیوتر**

**عنوان :**

**استفاده از ابزار "داکر" برای پیاده سازی مدل های یادگیری عمیق**

***استاد راهنما :***

**دکتر راضیه راستگو**

**دانشجو :**

**محمدنوید افضلی**

**شماره دانشجویی:**

**9711126012**

**نیمسال اول سال تحصیلی 02-01**

**فهرست مطالب**

[چکیده 7](#_Toc124168798)

[فصل اول 8](#_Toc124168799)

[مقدمه 8](#_Toc124168800)

[1-1 داکر چیست؟ 9](#_Toc124168801)

[1-1-1تاریخچه کوتاهی از داکر : 9](#_Toc124168802)

[1-1-2 نگهدارنده چیست؟ 10](#_Toc124168803)

[1-1-3 ماشین مجازی چیست؟ 11](#_Toc124168804)

[1-1-4 تفاوت داکر و ماشین مجازی در چیست؟ 11](#_Toc124168805)

[1-2 دلیل نیاز به استفاده از ابزار داکر چیست؟ 12](#_Toc124168806)

[1-3 کاربردهایداکر 14](#_Toc124168807)

[1-4 مزایا و معایب داکر 15](#_Toc124168808)

[1-4-1 مزایا: 15](#_Toc124168809)

[1-4-2 معایب: 15](#_Toc124168810)

[1-5 دلایل محبوبیت داکر 15](#_Toc124168811)

[1-6 اجزای اصلی داکر 16](#_Toc124168812)

[1-6-1 موتور داکر 16](#_Toc124168813)

[1-6-2 کلاینت داکر 16](#_Toc124168814)

[3-6-1 داکر دیمون 17](#_Toc124168815)

[5-6-1 داکر ایمیج 18](#_Toc124168816)

[6-6-1 داکر کانتینر 19](#_Toc124168817)

[7-6-1 داکر هاب 20](#_Toc124168818)

[8-6-1 سی گروپ ها 20](#_Toc124168819)

[9-6-1 نیم اسپیس ها 20](#_Toc124168820)

[7-1 مهم ترین دستورات داکر 21](#_Toc124168821)

[1-7-1 دستور docker info : 21](#_Toc124168822)

[3-7-1 دستور docker images : 22](#_Toc124168823)

[4-7-1 دستور docker pull : 23](#_Toc124168824)

[5-7-1 دستور docker push : 23](#_Toc124168825)

[6-7-1 دستور docker run : 23](#_Toc124168826)

[7-7-1 دستور docker build : 24](#_Toc124168827)

[8-7-1 دستور docker commit : 25](#_Toc124168828)

[فصل دوم 26](#_Toc124168829)

[داکر و یادگیری عمیق 26](#_Toc124168830)

[1-2 هوش مصنوعی چیست؟ 27](#_Toc124168831)

[2-2 یادگیری ماشین 28](#_Toc124168832)

[2-3 یادگیری عمیق 29](#_Toc124168833)

[2-4 تفاوت های یادگیری ماشین و یادگیری عمیق 30](#_Toc124168834)

[2-5 شبکه های عصبی 31](#_Toc124168835)

[1-5-2شبکه های عصبی کانولوشن 31](#_Toc124168836)

[7-2 کانولوشن 32](#_Toc124168837)

[8-2 انواع شبکه های عصبی کانولوشن 34](#_Toc124168838)

[9-2 کاربرد های شبکه های عصبی کانولوشن 34](#_Toc124168839)

[10-2 چرا از کانتینر های داکر در یادگیری عمیق استفاده میکنیم؟ 34](#_Toc124168840)

[1-10-2-کانتینرسازی 35](#_Toc124168841)

[فصل سوم 37](#_Toc124168842)

[پیاده سازی یادگیری ماشین(عمیق) با استفاده از داکر 37](#_Toc124168843)

[1-3 پروژه یادگیری ماشین 38](#_Toc124168844)

[1-1-3 مقدمات لازم برای ساخت ایمیج داکر 38](#_Toc124168845)

[2-1-3 شروع ساخت ایمیج 39](#_Toc124168846)

[3-1-3 اجرای ایمیج روی نگهدارنده 42](#_Toc124168847)

[3-2 پروژه یادگیری عمیق 44](#_Toc124168848)

[1-2-3 مقدمه 44](#_Toc124168849)

[2-2-3 شروع ساخت ایمیج 44](#_Toc124168850)

[3-3 جمع بندی و نتیجه گیری 50](#_Toc124168851)

[سخن پایانی 51](#_Toc124168852)

[منابع 52](#_Toc124168853)

[Abstract 53](#_Toc124168854)

**فهرست اشکال**

[**شکل 1-1 تفاوت عملکرد داکر و ماشین مجازی 11**](#_Toc123140818)

[**شکل 1-2 معماری داخلی داکر 13**](#_Toc123140819)

[**شکل 1-3 عملکرد داکر ؛ از ساخت نگهدارنده تا ذخیره سازی و استفاده از آن 13**](#_Toc123140820)

[**شکل 1-4 تعریف داکر رجیستری 17**](#_Toc123140821)

[**شکل 1-5 تعریف و نحوه عملکرد داکر ایمیج 18**](#_Toc123140822)

[**شکل 1-6 تعریف و نحوه عملکرد نگهدارنده داکر 19**](#_Toc123140823)

[**شکل 1-7 - دستور docker info 21**](#_Toc123140824)

[**شکل 1-8 - دستور docker ps 22**](#_Toc123140825)

[**شکل 1-9 - دستور docker images 22**](#_Toc123140826)

[**شکل 1-10 - دستور docker pull 23**](#_Toc123140827)

[**شکل 1-11 - دستور docker run 24**](#_Toc123140828)

[**شکل 2-1 نمودار ون هوش مصنوعی،یادگیری ماشین و یادگیری عمیق 28**](#_Toc123140829)

[**شکل 2-2 عمل کانولوشن 32**](#_Toc123140830)

[**شکل 2-3 پارامتر های اصلی کانولوشن 33**](#_Toc123140831)

[**شکل 3-1 داکرفایل 39**](#_Toc123140832)

[**شکل 3-2 مراحل ساخت ایمیج 41**](#_Toc123140833)

[**شکل 3-3 ایمیج های موجود در مخزن محلی (دریافت شده Docker hub ) 42**](#_Toc123140834)

[**شکل 3-4 اجرای ایمیج روی نگهدارنده 42**](#_Toc123140835)

[**شکل 3-5 محتویات dockerfile 45**](#_Toc123140836)

[**شکل 3-6 اجرای ایمیج2 46**](#_Toc123140837)

[**شکل 3-7 اجرای برنامه با کانتینر 47**](#_Toc123140838)

[**شکل 3-8 تصویر نمونه 48**](#_Toc123140839)

[**شکل 3-9 خروجی کانتینر تحت وب 49**](#_Toc123140840)

# **چکیده**

در طی سالیان اخیر و به ویژه امروزه ، از داکر به عنوان یکی از ابزار های مهم و کاربردی در حوزه برنامه نویسی یاد می­شود. داکر به عنوان ابزاری برای ساخت و اجرای برنامه ها در محیط های کانتینری باعث افزایش سرعت و بهره وری قابل توجه این برنامه ها گردیده است. با ابزار داکر دیگر نگرانی بابت تفاوت نسخه های متعلقات برنامه و کتابخانه های آن نخواهیم بود. هم اکنون از این ابزار در توسعه ی برنامه های مربوط به هوش مصنوعی و یادگیری ماشین نیز قابل استفاده است و توسعه دهندگان این حوزه می­توانند برنامه های خود را بدون دغدغه از طریق داکر با همکاران خود به اشتراک بگذارند.

در این پروژه قصد داریم با استفاده از ابزار های موجود در داکر ، یک نگهدارنده[[1]](#footnote-1) روی مدل یادگیری ماشین/عمیق [[2]](#footnote-2)خود ساخته و پیاده سازی کنیم. در واقع ایده ی این پروژه ساخت سریع و آسان نگهدارنده داکر با یک مدل یادگیری عمیق ساده و اجرای آن است.

# فصل اول

# مقدمه

# 1-1 داکر***[[3]](#footnote-3)*** چیست؟

داکر پروژه ای متن باز[[4]](#footnote-4) برای توسعه و اجرای برنامه ها بر مبنای نگهدارنده ها یا Container می­باشد و عملیات ساخت[[5]](#footnote-5) ، اجرا[[6]](#footnote-6) و مدیریت برنامه را تسهیل می­ بخشد. داکر می‌تواند یک اپلیکیشن و متعلقات آن را(کتابخانه ها و ...) در یک نگهدارنده مجازی اجرا کند. این ابزار توسط زبان برنامه نویسی GO توسعه یافته است.[6]

## 1-1-1تاریخچه کوتاهی از داکر :

* **سال ۲۰۰۶:**در این سال پروسس کانتینر[[7]](#footnote-7) ارائه شد.
* **سال ۲۰۰۸:** در این سال [[8]](#footnote-8)LXC ارائه شد که این موضوع مقدمه‌ای مناسب و خوبی برای استفاده از کانتینرها بود.
* **سال ۲۰۱۳:** سرویس داکر در این سال ارائه شد. در همین سال شرکت بزرگ رد هت[[9]](#footnote-9) اعلام کرد که پروژه‌های لینوکس[[10]](#footnote-10)،اوپن شیفت[[11]](#footnote-11) و فدورا[[12]](#footnote-12) را به سمت این پروژه‌ خواهد برد و از این پروژه حمایت کرد.
* **سال ۲۰۱۴:**در این سال شرکت ماکروسافت اعلام کرد که نسخه‌ی آینده ویندوز سرور خودش رو به سمتی خواهد برد که بتواند داکر کلاینت[[13]](#footnote-13) رو پشتیبانی کند. شرکت IBM نیز در همین سال اعلام کرد که برنامه‌ریزی خواهد کرد تا در IBM Cloud از این سرویس استفاده خواهد کرد و در همین سال شراکت خودش را با سرویس داکر اعلام کرد.
* **سال ۲۰۱۶:**شرکت داکر سرویس تجاری[[14]](#footnote-14) خود را در این سال ارائه کرد.
* **سال ۲۰۱۸:**در حال حاضر سرویس داکر دارای جامعه‌ی متن باز بسیار پویا و کاملی می‌باشد که بسیار فعال بوده و تقریبا همواره در حال رشد و شکوفایی بیشتر می‌باشد[4].

در ادامه به تعریف نگهدارنده و ماشین مجازی می­پردازیم.

## 1-1-2 نگهدارنده چیست؟

نگهدارنده[[15]](#footnote-15) یک واحد نرم‌افزاری استاندارد است که کدها و تمام متعلقات آن را بسته‌بندی می­کند. به این ترتیب، اپلیکیشن در محیط‌های محاسباتی مختلف، سریع‌تر و با اطمینان بیش‌تر اجرا می‌شود. هر کانتینر یک محیط ایزوله شده را مشابه یک ماشین مجازی[[16]](#footnote-16) فراهم می‌کند. برخلاف ماشین‌های مجازی، کانتینرهای داکر یک سیستم عامل کامل را اجرا نمی‌کنند، بلکه هسته[[17]](#footnote-17) میزبان را به اشتراک می‌گذارند و مجازی‌سازی را در یک سطح نرم‌افزاری انجام می‌دهند.

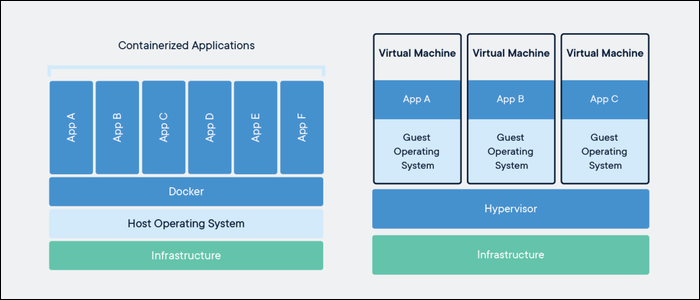
برای درک بهتر نگهدارنده ها، بهتر است از یک مثال استفاده کنیم. فرض کنید در شرکتی مشغول به کار هستید و ناهار خود را هر روز در خانه درست کرده و آن را داخل یک ظرف به شرکت می‌برید تا آنجا میل بفرمایید. دیگر لازم نیست داخل شرکت شروع به پختن غذا کنید چون احتمالا زمان زیادی را از شما می‌گیرد. کار نگهدارنده هم تا حدودی شبیه به این است. شما پروژه خود را (غذا) داخل نگهدارنده (ظرف غذا) قرار داده و آن را هر کجا که دوست داشتید (مثلا شرکت) می‌برید.

به لطف این تکنیک، برنامه نویس خیالش راحت است که برنامه او می‌تواند در سیستم‌های دیگر بدون نیاز به تنظیمات خاص یا ابزارهای جانبی اجرا شود.

## 1-1-3 ماشین مجازی چیست؟

ماشین مجازی برنامه‌ای است که به عنوان یک رایانه مجازی عمل می‌کند. این برنامه بر روی سیستم‌عامل فعلی شما اجرا می‌شود و سخت‌افزاری مجازی برای یک سیستم‌عامل میهمان ارائه می‌کند. از نگاه سیستم عامل میهمان ، ماشین مجازی مشابه یک رایانه واقعی عمل می­کند. ماشین های مجازی از سخت افزار های سیستم همچون CPU ، حافظه RAM و سایر سخت افزار ها استفاده می­کند. البته نحوه عملکرد داکر با ماشین مجازی متفاوت است که در ادامه شرح داده می شود.

1-1-4 تفاوت داکر و ماشین مجازی در چیست؟

****

شکل 1-2 تفاوت عملکرد داکر و ماشین مجازی

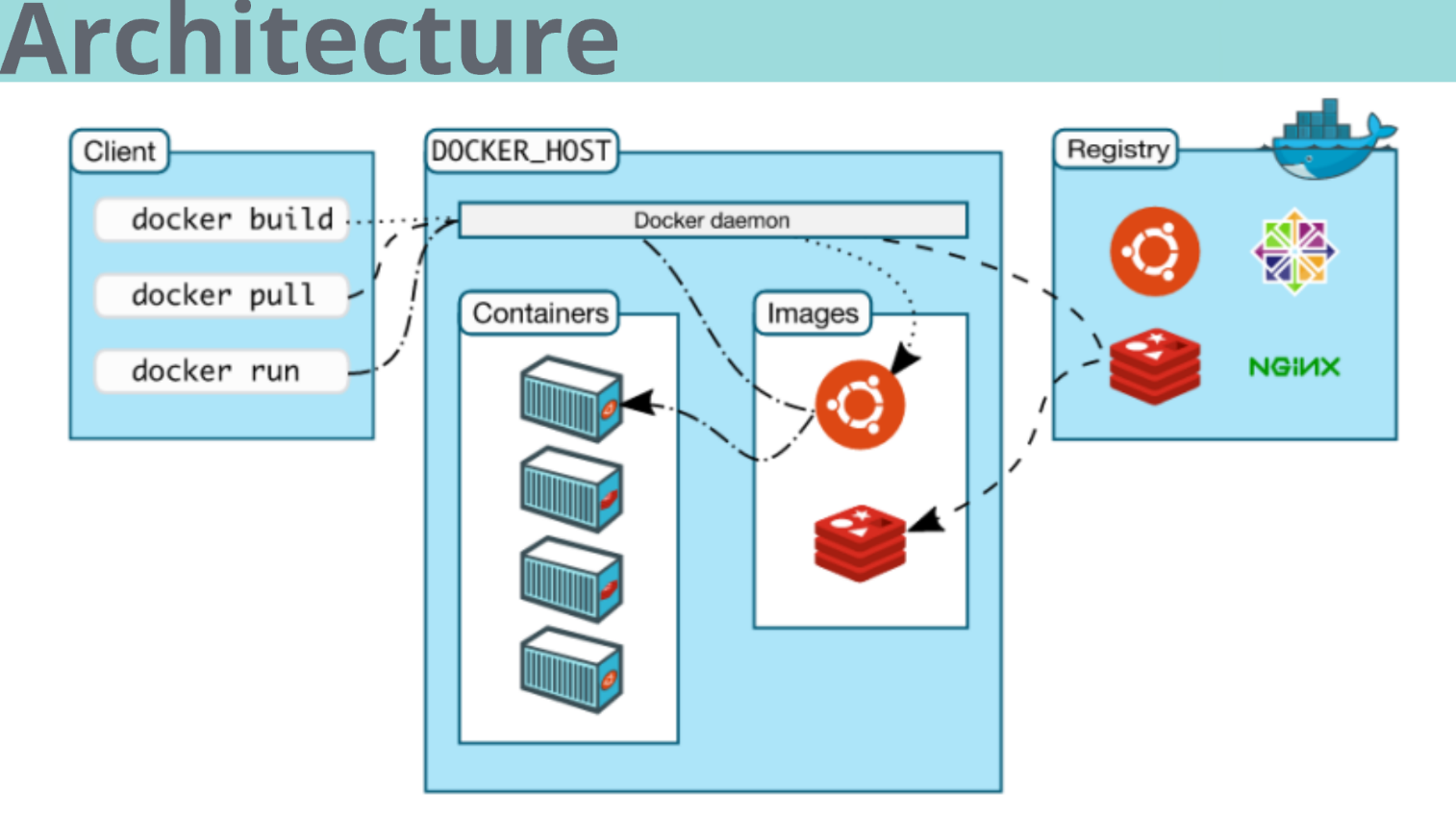
در ماشین مجازی، قسمتی از سخت‌افزار سیستم به ماشین مجازی اختصاص داده می‌شود و روی آن یک سیستم‌عامل کامل نصب و اجرا می‌شود. در واقع می‌توان گفت در ماشین مجازی امکانات سخت‌افزاری سیستم تقسیم می‌شود و بر روی هر قسمت، سیستم‌عامل بخصوصی به اجرا در می آید ، اما در داکر این طور نیست. در داکر امکانات سخت‌افزاری به تناسب نیاز هر کانتینر به صورت موقت اختصاص داده می‌شود و داکر این امکان را فراهم می‌آورد که اپلیکیشن‌ها روی کرنل سیستم عامل اجرا شوند.این عمل ، بازدهی و کارایی سیستم را تا حد زیادی بالا می­برد. در این حالت دیگر نیازی به نصب پیش‌نیازها و نیازمندی‌هایی که برنامه ی ما می‌خواهد و به طور پیش‌فرض روی سیستم وجود ندارد، نیست.

این سرویس به شما کمک می‌کند یک محیط را به چند بخش تقسیم کرده و و در هر بخش یک برنامه مجزا اجرا کنید.

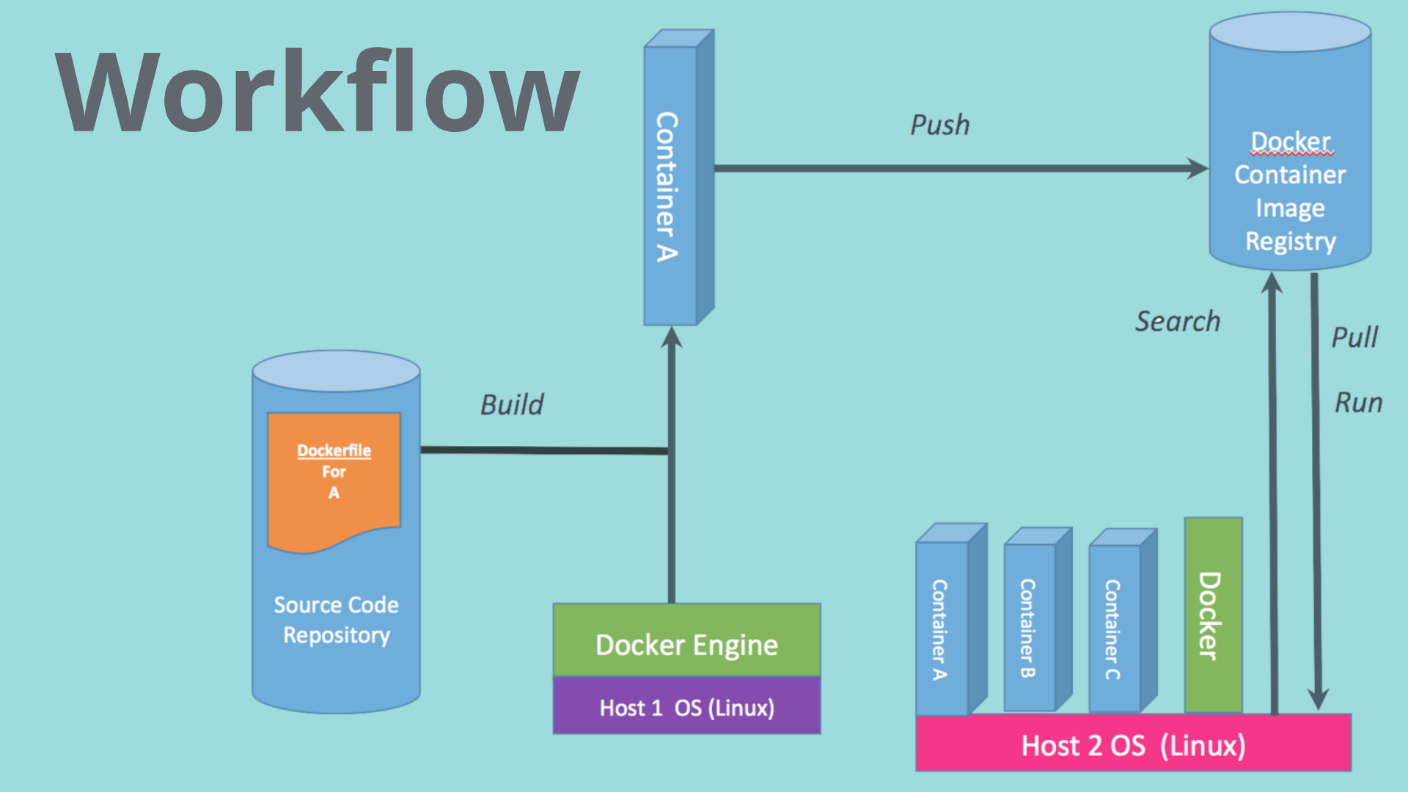
در ادامه به دلایل نیاز به استفاده از داکر می­پردازیم.

# 1-2 دلیل نیاز به استفاده از ابزار داکر چیست؟

با ذکر یک مثال ، دلیل نیاز به این ابزار را نشان می­دهیم. برای مثال، سه اپلیکیشن مختلف مبتنی بر پایتون[[18]](#footnote-18)  وجود دارد که قرار است روی یک سرور میزبانی شوند (چه فیزیکی چه مجازی). همچنین، فرض می‌کنیم که در هر یک از این اپلیکیشن‌ها ورژن های متفاوتی از پایتون استفاده شده و کتابخانه‌ها و متعلقات به کار گرفته شده نیز در هر اپلیکیشن با دیگری فرق دارد. با توجه به اینکه نمی‌توان نسخه‌های مختلفی از پایتون را روی یک ماشین نصب کرد، امکان میزبانی از این سه اپلیکیشن روی یک کامپیوتر واحد وجود نخواهد داشت. اینجاست که فناوری داکر به کار می‌آید. دیگر نیاز نیست که چند ماشین مجازی مختلف را برای نسخه های مختلف پایتون اجرا کنیم و قسمت زیادی از سخت افزار خود را از دست دهیم. تنها یک بار داکر را نصب می­کنیم؛ داکر ، خود نگهدارنده های مجزا را ایجاد می­کند. بدین ترتیب یک محیط را به چند بخش تقسیم کرده و در هر قسمت ، یک برنامه مجزا اجرا می­کنیم.

****

شکل 1-3 معماری داخلی داکر

****

شکل 1-4 عملکرد داکر ؛ از ساخت نگهدارنده تا ذخیره سازی و استفاده از آن

# **1-3** کاربرد های داکر [6]

* **تولید اپلیکیشن های سریع و پایدار**

داکر منجر به ساده‌سازی چرخه حیات توسعه می‌شود. این ساده‌سازی به وسیله فراهم کردن امکان کار در محیط‌های استاندارد‌سازی شده برای توسعه‌دهندگان با استفاده از نگهدارنده های محلی فراهم شده است که اپلیکیشن‌ها و خدمات لازم را ارائه می‌دهند.

* **اجرای حجم کاری بیشتر بدون نیاز به ارتقای سخت افزار**

داکر کم‌حجم و سریع است .داکر جایگزینی مقرون به صرفه برای ماشین‌های مجازی مبتنی بر لایه نرم‌افزاری[[19]](#footnote-19) به حساب می‌آید. با داکر می‌توان ظرفیت محاسباتی بیش‌تری را برای دست‌یابی به اهداف کسب و کار به کار گرفت. داکر برای محیط‌هایی با تراکم بالا و همچنین برای استقرار نرم‌افزارهای کوچک و متوسط یعنی جایی مناسب است که نیاز به اجرای وظایف بیش‌تر با منابع کم‌تر وجود داشته باشد.

* **مقیاس پذیری**

نگهدارنده در داکر امکان ایجاد حجم کاری[[20]](#footnote-20) قابل حمل را فراهم می‌سازد. نگهدارنده های داکر می‌توانند روی لپ‌تاپ توسعه‌دهنده، ماشین‌های فیزیکی یا مجازی در یک مرکز داده، ارائه دهندگان فضای ابری یا روی محیط‌های تلفیقی اجرا شوند.

# 1-4 مزایا و معایب داکر

در ادامه به مزایا و معایب داکر می­پردازیم: [6]

## 1-4-1 مزایا:

* ثبات و پایداری
* صرفه جویی در فضای ذخیره سازی
* وسعت و انعطاف جامعه توسعه دهندگان
* امکان استفاده در مک و ویندوز
* خودکارسازی

## 1-4-2 معایب:

* کمبود در مستندات
* در حال حاضر مشکلات کارکردی برای Mac وجود دارد
* بروز مشکلات در محیط های غیر بومی

# 1-5 دلایل محبوبیت داکر

داکر به دلیل راه‌اندازی بر بستر کانتینر کیفیت برنامه‌ها را بالاتر برده و همچنین به دلیل قابل حمل بودن آن، شرایط استفاده در هر مکانی را داراست.

امکان محفظه‌سازی تنها برای یک بار و اجرا در همه محیط‌ها، منجر به کاهش فاصله میان محیط توسعه و سرورهای تولید محصول می‌شود. استفاده از نگهدارنده ها این اطمینان و اعتماد را به وجود می‌آورد که همه محیط‌ها برابر هستند. در صورتی که عضو جدیدی به تیم توسعه اضافه شود، این فرد تنها لازم است از دستور «docker run» برای راه‌اندازی نمونه توسعه خود استفاده کند(این دستورات در ادامه معرفی خواهند شد).

داکر نسبت به یک ماشین مجازی بسیار ساده‌تر است. ماشین‌های مجازی ابزارهایی همه‌منظوره هستند که برای پشتیبانی از هر میزان حجم کاری ممکن طراحی شده‌اند. در مقابل آن، نگهدارنده های کم حجم داکر، خودکفا و برای کاربردهای یک بار مصرف مناسب‌تر هستند. با توجه به اینکه داکر کرنل میزبان را به اشتراک می‌گذارد، نگهدارنده ها تاثیر قابل چشم‌پوشی در کاهش عملکرد و کارایی سیستم دارند. زمان اجرای نگهدارنده ها تقریباً لحظه‌ای و بلافصل[[21]](#footnote-21) است.

# **1-6 اجزای اصلی داکر**

## **1-6-1 موتور داکر[[22]](#footnote-22)**

موتور داکر هسته شروع تمام کار ها در این ابزار است. تا زمانی که موتور داکر آغاز به کار نکند استفاده از داکر ممکن نیست. این موتور حکم اتصال اینترنت را برای این ابزار دارد. در سیستم عامل های ویندوز این موتور به وسیله برنامه docker desktop آغاز به کار می­کند.

## **1-6-2 کلاینت داکر[[23]](#footnote-23)**

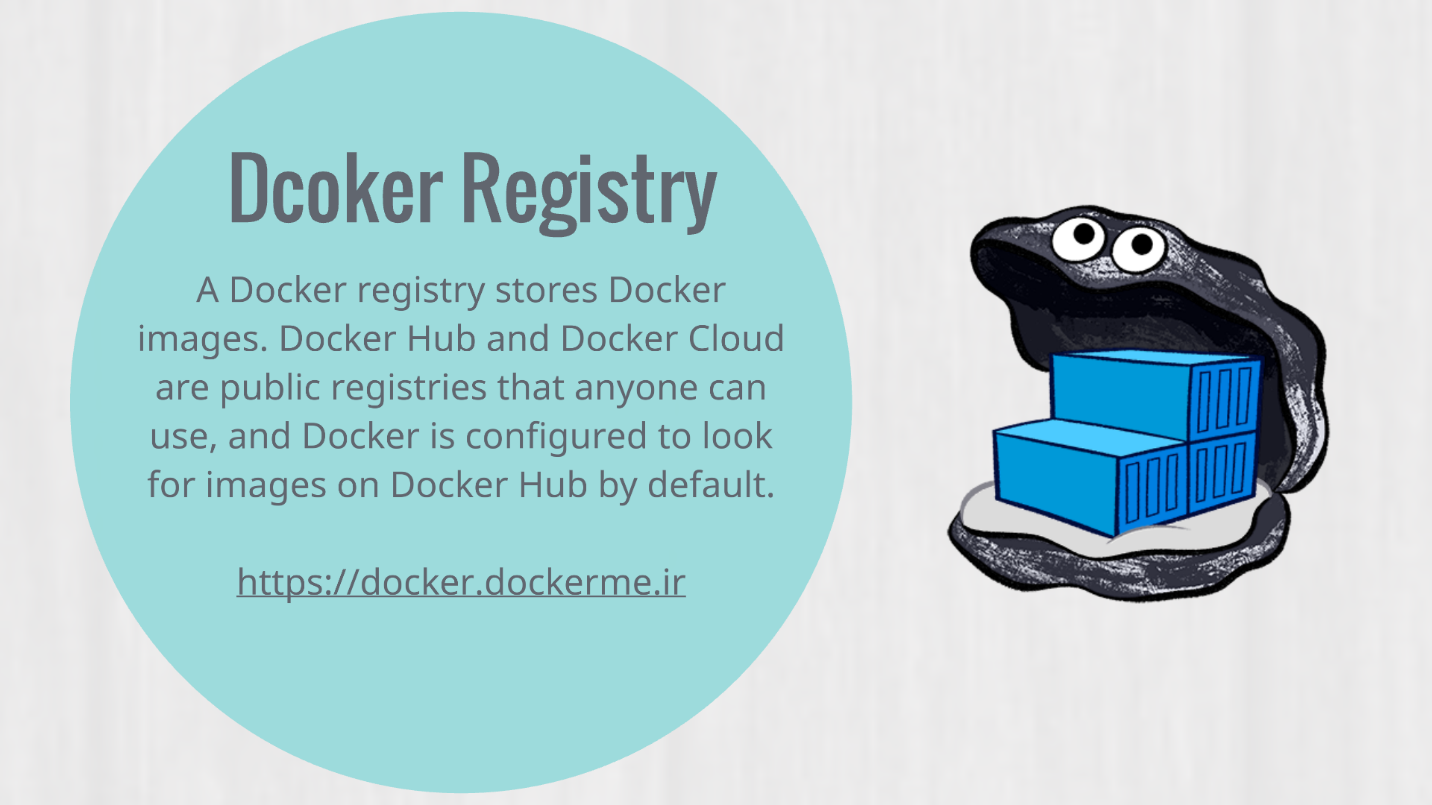
کلاینت داکر جزئی ست که کاربر نهایی با آن در ارتباط خواهد بود. برای درک بهتر می­توانید آن را یک [[24]](#footnote-24)UI برای داکر بنامید. هر چند که فاصله آن از واژه کاربرپسند[[25]](#footnote-25) بسیار زیاد است. شما به عنوان کاربر نهایی با کلاینت داکر ارتباط خواهید داشت و کلاینت داکر دستورات شما را به داکر دیمون منتقل می نماید.

## **3-6-1 داکر دیمون[[26]](#footnote-26)**

داکر دیمون جزئی ست که دستورات ارسال شده به کلاینت داکر را اجرا می­نماید. دستوراتی مانند ساختن ، راه اندازی و یا توزیع نگهدارنده ها .

داکر دیمون ، خود ، بر روی هاست اجرا می­شود. اما به عنوان کاربر شما هیچ گاه به صورت مستقیم با آن در ارتباط نخواهید بود. همان طور که در قسمت قبل در توضیح کلاینت داکر گفته شد ، کلاینت داکر انتقال دهنده دستورات شما به داکر دیمون خواهد بود.

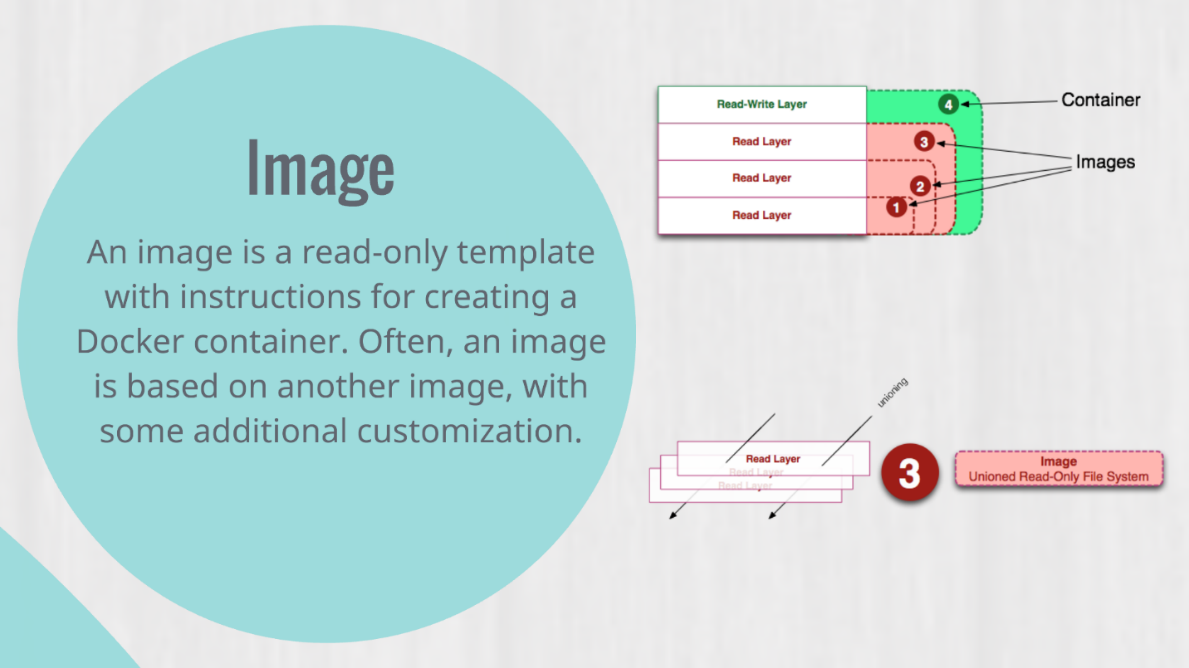
**4-6-1 داکر رجیستری[[27]](#footnote-27)**

****

شکل 1-5 تعریف داکر رجیستری؛ [4]

داکر رجیستری فضای ذخیره سازی ایمیج‎های داکری می‎باشد . (ایمیج های داکر در ادامه توضیح داده خواهد شد) . مخزن Docker Hub یک مخزن عمومی داکر است که تمام کاربران می­توانند ایمیج هایی که برای برنامه خود ایجاد می کنند را در آن ذخیره کنند تا دیگران هم بتوانند از آن استفاده کنند.

## **5-6-1 داکر ایمیج[[28]](#footnote-28)**

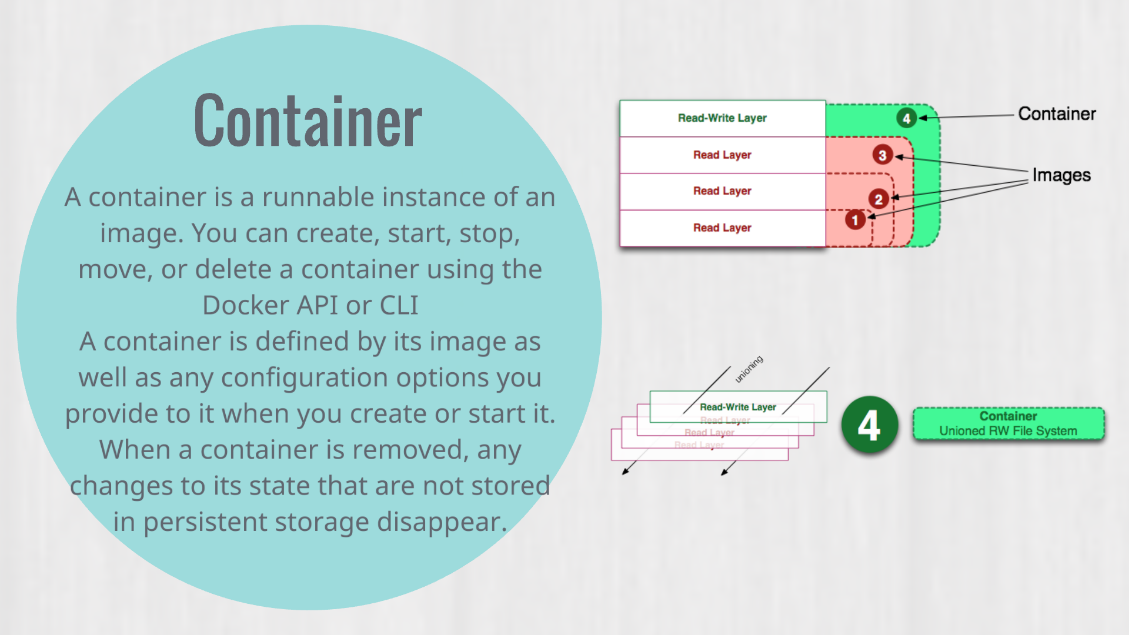
****

شکل 1-6 تعریف و نحوه عملکرد داکر ایمیج ؛ [4]

داکر ایمج یک فایل غیر قابل تغییر است که شامل کد منبع، کتابخانه ها[[29]](#footnote-29)، وابستگی ها ، ابزارها و سایر فایل های مورد نیاز برای اجرای یک برنامه است.

از آنجا که ایمیج ها به نوعی فقط الگو یا قالب[[30]](#footnote-30) هستند، نمی توانید آن ها را استارت یا اجرا کنید. کاری که می توانید انجام دهید این است که از آن الگو به عنوان پایه برای ساخت نگهدارنده استفاده کنید.

## **6-6-1 داکر کانتینر**

****

شکل 1-7 تعریف و نحوه عملکرد نگهدارنده داکر؛[4]

کانتینر داکر یک محیط مجازی run-time  است که در آن کاربران می توانند برنامه ها را از سیستم اصلی جدا کنند. این نگهدارنده ها واحد های قابل حمل و فشرده ای هستند که می توانید در آن ها به راحتی یک برنامه را راه اندازی کنید. یک ویژگی مهم دیگر، استاندارد‌ سازی محیط محاسبات و رایانش در حال اجرا در داخل نگهدارنده است. این نه تنها از این که برنامه شما در شرایط یکسانی در حال کار است اطمینان حاصل می کند، بلکه اشتراک گذاری با سایر هم تیمی ها را نیز ساده می­سازد.

## **7-6-1 داکر هاب[[31]](#footnote-31)**

داکر هاب سرور ذخیره ساز [[32]](#footnote-32) مبتنی بر ابر است که مصرف کنندگان وکاربران داکر می­توانند در آن ایمج های کانتینر را ایجاد، تست، ذخیره سازی و توزیع کنند.

**داکر هاب  نسخه ای از داکر رجیستری می باشد که میزبانی آن مبتنی بر ابر**[[33]](#footnote-33)**است.**

## **8-6-1 سی گروپ ها[[34]](#footnote-34)**

سی گروپ ها اجازه تخصیص منابع را به ما می­دهند. منظور از منابع ، به طور کلی سخت افزار که شامل سی پی یو[[35]](#footnote-35)، حافظه ، شبکه ، دیسک ورودی و خروجی[[36]](#footnote-36) و ... می­باشد. منابع در داکر توسط سی گروپ ها مدیریت می­شوند.

سی گروپ ها به موتور داکر [[37]](#footnote-37)اجازه می­دهند تا منابع سخت افزاری موجود را به نگهدارنده ها اختصاص دهند و به صورت اختیاری می­توانند محدودیت های لازم را به اجرا در آورند؛ برای مثال ، شما می­توانید حافظه ی موجود را برای نگهدارنده ی خاصی محدود کنید.

## **9-6-1 نیم اسپیس ها[[38]](#footnote-38)**

داکر از namespace ها برای تهیه فضاهای مجزا به نام نگهدارنده استفاده می­کند.

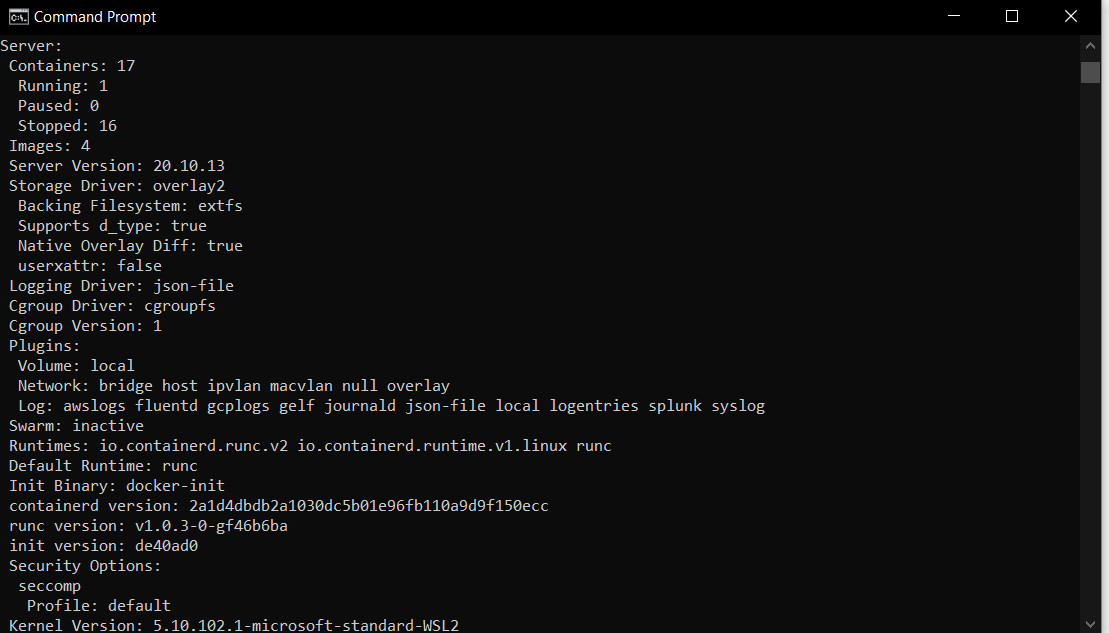
از طرف دیگر namespaces مکانیزم دیگری در کرنل برای محدود کردن دید گرهی از فرآیند ها، نسبت به بقیه سیستم هاست. برای مثال می­تواند یک یا چند فرآیند را محدود کرد تا امکان دیدن و تعامل با بقیه فرآیند های در حال اجرا روی سیستم یا مثلا امکان دسترسی به فایل سیستم های مانت [[39]](#footnote-39)شده را نداشته باشد.

# **7-1 مهم ترین دستورات داکر**

در ادامه به برخی از مهم ترین دستورات داکر که از آن ها به صورت متداول در خط فرمان[[40]](#footnote-40) استفاده می شود، می­پردازیم:

## **1-7-1 دستور docker info :**

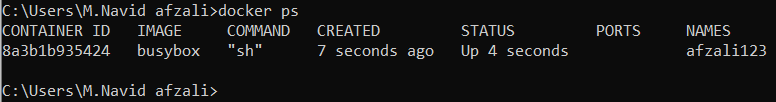
این دستور توضیح کامل و جامعی از سرویس داکری که بر روی سیستم ما قرار دارد ، در اختیار ما قرار می­دهد. اطلاعاتی مانند تعداد نگهدارنده ها و وضعیت هر کدام از آن ها، تعداد ایمیج ها ، نسخه ی کرنل سیستم عامل ، میزان منابعی که در اختیار دارید و ..

****

شکل 1-8 - دستور docker info

**2-7-1 دستور docker ps :**

این دستور لیست نگهدارنده ها روی سرویس دهنده داکر شما را ارائه می­دهد . شامل تنظیمات[[41]](#footnote-41) بسیاری می­باشد . در حالت کلی و بدون تنظیمات ، فقط نگهدارنده های در حال کار[[42]](#footnote-42) را نمایش می­دهد.

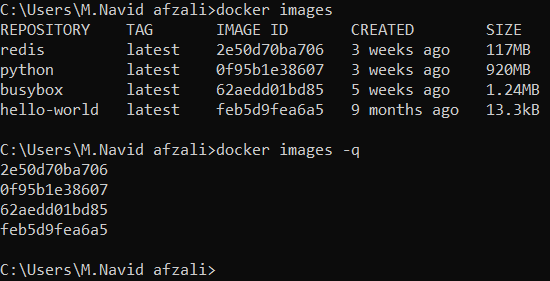
****

شکل 1-9 - دستور docker ps

## 

## **3-7-1 دستور docker images :**

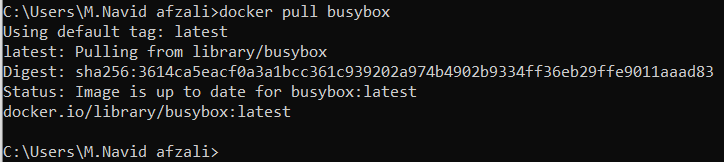
این دستور لیست ایمیج های ما را نمایش می­دهد . همانند دستور قبل شامل تنظیمات زیادی است . برای مثال ، با تنظیم -q تنها شناسه ایمیج[[43]](#footnote-43) های موجود را برمی­گرداند.

****

شکل 1-10 - دستور docker images

## **4-7-1 دستور docker pull :**

توسط این دستور ، شما ایمیج های مورد نیاز خود را از رجیستری های مختلف ( به طور پیش فرض hub.docker.com ) دانلود و دریافت می­کنید . تنظیمات خاصی ندارد و برای دریافت ایمیج مورد نیاز خود ، نام ایمیج و برچسب[[44]](#footnote-44) مورد نظر را وارد می­کنید. اگر برچسبی وارد نکنید ، داکر به صورت پیش فرض برچسب latest را در نظر می­گیرد.

****

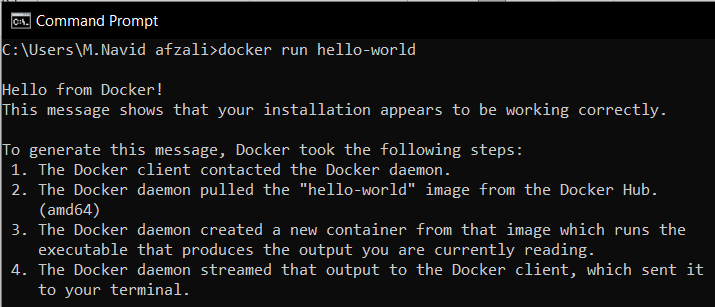
شکل 1-11 - دستور docker pull

## **5-7-1 دستور docker push :**

برعکس دستور قبلی عمل می­کند . به طوری که با این دستور می­توانید ایمیج خود را در داخل رجیستری های مختلف داکر قرار دهید. تنظیمات خاصی ندارد ، فقط باید توجه داشت که برای قرار دادن ایمیج درون رجیستری ، باید از قبل وارد[[45]](#footnote-45) یک رجیستری شده باشید.

## **6-7-1 دستور docker run :**

به وسیله ی این دستور می­توانید از روی ایمیج های موجود در سرویس داکر ، یک نگهدارنده راه اندازی کرد. دارای تنظیمات بسیار زیادی می­باشد که توضیح آنها از حوصله ی این پروژه خارج است.(دستورات مهم در ادامه پروژه شرح داده خواهند شد)

****

شکل 1-12 - دستور docker run

## **7-7-1 دستور docker build :**

به وسیله ی این دستور، قادر خواهیم بود از روی داکر فایل[[46]](#footnote-46) ، ایمیج داکر ایجاد کنیم. فرآیند build به این صورت است که از روی یک ایمیج پایه[[47]](#footnote-47) مابقی تغییرات در آن داده می‌شود تا ایمیج مد نظر شما ایجاد شود. نکته این که در این مسیر ممکن چندین ایمیج ایجاد شود تا به ایمیج نهایی برسیم. این قابلیت لایه‌ای بودن بسیار به کم‌ حجم شدن ایمیج نهایی می‌تواند کمک کند. ممکن است در لایه‌ی آخر ایمیج شما مشکلی رخ دهد که مجبور باشید بعد از اصلاح آن از ابتدا فرآیند build را انجام دهید. قابلیت کش[[48]](#footnote-48) که از همین روش لایه‌ای بودن ایمیج‌ها فراهم شده است به شما کمک می‌کند که تمام لایه‌های درست و صحیح را مجدد دریافت نکنید و تنها از همان لایه‌ای که خطا داشته است ادامه دهید.

## **8-7-1 دستور docker commit :**

به کمک این دستور ، می­توانیم از نگهدارنده ی در حال اجرا ، یک ایمیج ایجاد کنیم ؛ این دستور زمانی موثر است که تغییراتی که در نگهدارنده ایجاد کرده­ایم ، مهم است و در آینده بار های به آن نیاز می شود. پس مطلوب است که این تغییرات در ایمیج جدیدی ثبت شود.

دستوراتی که تاکنون ذکر شدند دستورات اصلی داکر هستند. داکر شامل دستورات زیادی است که مابقی آن ها دستورات پیشرفته داکر محسوب می­شوند و فراتر از این پروژه هستند. بنابراین به معرفی این دسته از دستورات نمی­پردازیم.

فصل اول در اینجا به پایان می­رسد. در فصل دوم به معرفی هوش مصنوعی و توضیحاتی پیرامون یادگیری ماشین و یادگیری عمیق و کاربرد داکر در آنها خواهیم پرداخت.

# فصل دوم

# داکر و یادگیری عمیق

# **1-2 هوش مصنوعی چیست؟**

هوش مصنوعی به هوشمندی گفته می شود که از ماشین ها حاصل می­شود. به عبارتی، هر ماشینی که می­تواند محیط را درک کند و فعالیت های وابسته به فکر و ذهن انسان را به خوبی انجام دهد.[11]

**آلن تورینگ** یکی از تاثیرگذارترین افراد در حوزه هوش مصنوعی است و بسیاری مقاله‌ای که او در سال ۱۹۵۰ منتشر کرد را تولد هوش مصنوعی می‌دانند. او همچنین تست تورینگ[[49]](#footnote-49) را پیشنهاد داد که معیاری برای تشخیص هوشمندی ماشین است.

یکی از اولین پروژه‌های عملی هوش مصنوعی، ایده گرفتن از مفهوم **شبکه عصبی** و نورون‌ها بود. دو نفر از دانشجویان دانشگاه هاروارد در سال ۱۹۵۰ اولین شبکه عصبی مصنوعی را که شامل ۴۰ نورون بود ساختند. نورون‌ها واحد‌هایی یک‌شکل در مغز هستند که پیام‌های عصبی را منتقل می‌کنند. جالب است بدانید شبکه‌های عصبی مصنوعی جدید که از آن‌ها برای طبقه‌بندی تصاویر استفاده می‌شود دارای هزاران نورون هستند!

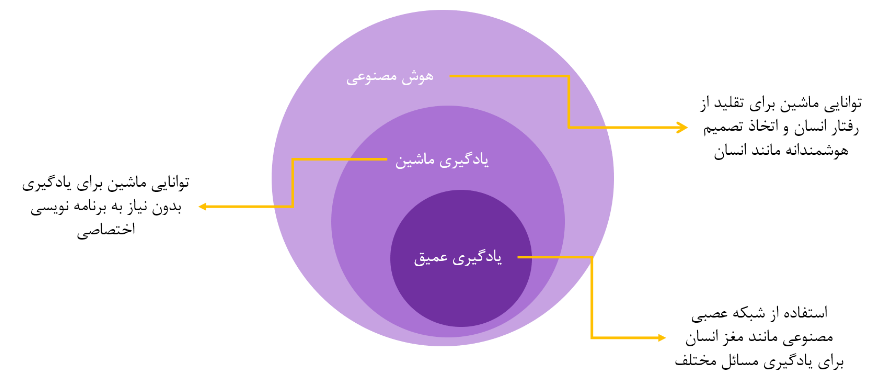
باخت کاسپاروف، قهرمان شطرنج جهان، از سیستم deep blue  باعث شد تا هوش‌ مصنوعی در کانون توجه قرار گیرد. این سیستم که توسط IBM  طراحی شده بود؛ در اولین مسابقه در سال ۱۹۹۶ کاسپاروف پیروز شد اما در رقابت بعدی که در سال ۱۹۹۷ اتفاق افتاد، deep blue  توانست کاسپاروف را شکست دهد.

این حوزه دارای شاخه هایی است که در زمینه های مختلف نیز مورد استفاده قرار می­گیرند. در ادامه ما یادگیری ماشین را به عنوان زیرمجموعه ای از هوش مصنوعی بررسی می­کنیم.

# **2-2 یادگیری ماشین[[50]](#footnote-50)**

یادگیری ماشین بعد از هوش مصنوعی موضوعیت پیدا کرد و آن را می‌توان زیر مجموعه هوش مصنوعی در نظر گرفت. یادگیری ماشین ویژگی‌هایی دارد که آن را از دیگر روش‌های هوش مصنوعی متمایز می‌کند.

یکی از مهم‌ترین ویژگی‌های یادگیری ماشین، یادگیری با استفاده از داده و بدون استفاده از دستورات مستقیم است. در انواع دیگر هوش مصنوعی ممکن است شاهد نوعی از هوشمندی باشیم که از داده‌ها استفاده نمی‌کند. ویژگی دیگر یادگیری ماشین، تغییر خود با دیدن داده‌های جدید است؛ یعنی همانند انسان با کسب تجربیات جدید رفتار خود را تغییر می‌دهد[7].



شکل 2-1 نمودار ون مربوط به هوش مصنوعی،یادگیری ماشین و یادگیری عمیق

در یادگیری ماشین به طراحی ماشین هایی پرداخته می­شود که با توجه به دیتاهای داده شده به آن ماشین و تجربیات ماشین، بیاموزند. به عبارتی ؛ یادگیری ماشین کامپیوتر ها را قادر می­سازد بدون این که صریحا برنامه­ریزی شده باشند، به روشی خودآموز تبدیل شوند.

یک برنامه یادگیرنده برنامه رایانه‌ای است که به آن گفته شده تا از تجربه E مطابق با برخی وظایف T، و کارایی عملکرد P برای وظیفه T که توسط P سنجیده می‌شود، یاد بگیرد که تجربه E را بهبود ببخشد.(تام میشل)

یافتن الگوها در داده‌های موجود در سیاره زمین، تنها برای مغز انسان ممکن است. اما هنگامی که حجم داده‌ها بسیار زیاد می‌شود و زمان لازم برای انجام محاسبات افزایش می‌یابد، نیاز به یادگیری ماشین به عنوان علمی مطرح می‌شود که به افراد در کار با داده‌های انبوه در حداقل زمان کمک می‌کند.(فرادرس) امروزه یادگیری ماشینی در بسیاری زمینه‌ها از جمله مهندسی، کسب و کار، زبان‌شناسی و پزشکی کاربرد دارد.

استفاده از یادگیری‌ماشین به دلایل زیر خیلی آسان‌تر و فراگیرتر شده است:

* افزایش بی حد و حصر داده‌ها
* ذخیره سازی مقرون به صرفه داده‌ها
* افزایش قدرت پردازنده‌ها و کاهش هزینه‌های آن‌ها

در ادامه به توضیح مختصر یادگیری عمیق، یکی از مهم ترین زیرمجموعه های یادگیری ماشین می­پردازیم:

# **2-3 یادگیری عمیق**

شاید در مورد یادگیری عمیق هم چیزهایی شنیده باشید. یادگیری عمیق عمری بسیار کوتاه‌تر از یادگیری ماشین دارد. استفاده از شبکه‌های عصبی مصنوعی یکی از انواع روش‌های یادگیری ماشین است که از سال‌ها پیش در حال گسترش است. این شبکه‌ها از ساختار ذهن انسان الگوبرداری شده‌اند و ساختاری لایه لایه دارند. با پیشرفت تکنولوژی و افزایش توان محاسباتی کامپیوتر‌ها، مهندسان توانستند تعداد لایه‌های این شبکه‌ها را افزایش دهند و این شروعی برای یادگیری عمیق بود. یادگیری عمیق می‌تواند الگوهای پیچیده‌تری را در داده پیدا کند و امروزه به یکی از جذاب‌ترین بخش‌های یادگیری ماشین تبدیل شده است[2].

یادگیری عمیق اشاره به مجموعه ای از الگوریتم های یادگیری ماشین دارد, که معمولا مبتنی بر شبکه های عصبی مصنوعی اند و تلاش دارند تا انتزاعات سطح بالای موجود در داده ها را مدل نمایند. یادگیری عمیق دقت بالاتری نسبت به سایر مدل های یادگیری ماشین دارا می­باشد.

یادگیری عمیق کاربرد های زیادی دارد همچون : بازسازی تصاویر ، حذف نویز در تصاویر ، تشخیص گفتار ، تبدیل متن به گفتار و ...

# **2-4 تفاوت های یادگیری ماشین و یادگیری عمیق [2]**

* استخراج ویژگی ها : استخراج ویژگی ها در یادگیری ماشین به صورت دستی انجام می­شود؛ در حالی که در یادگیری عمیق به صورت خودکار انجام می­پذیرد.
* دقت: یادگیری عمیق دقت بالاتری نسبت به یادگیری ماشین دارد.
* زمان آموزش : یادگیری ماشین زمان کوتاه تری برای آموزش نیاز دارد؛ در حالی که در یادگیری عمیق این زمان طولانی تر است.
* نیازمندی داده : یادگیری عمیق نیازمند داده های زیاد و کلان است؛ یادگیری ماشین با داده های کم نیز قابل آموزش است.
* شکل پیاده سازی : یادگیری عمیق از شبکه های عصبی استفاده می­کند؛ درحالی که یادگیری ماشین از الگوریتم هایی مانند خوشه بندی و رگرسیون بهره می­برد.

# **2-5 شبکه های عصبی[[51]](#footnote-51)**

وقتی به یادگیری عمیق که حیطه‌ای از یادگیری ماشین است، اشاره می‌شود، احتمالاً پای شبکه های عصبی در میان است. شبکه‌های عصبی از مغز ما الگو گرفته‌اند. گره[[52]](#footnote-52) هایی وجود دارند که لایه‌ها را در شبکه تشکیل می‌دهند و دقیقاً مانند نورون‌های مغز ما، نواحی مختلف را به هم متصل می‌کنند. به ورودی‌های نود‌ها در یک لایه، وزنی اختصاص می‌یابد که تأثیری را که پارامتر بر نتیجه پیش‌بینی کلی دارد، تغییر می‌دهد. از آنجا که وزن‌ها به پیوندهای بین نود‌ها اختصاص داده می‌شوند، ممکن است هر نود تحت‌تأثیر وزن‌های مختلف قرار گیرد[6].

# **1-5-2 شبکه های عصبی کانولوشن[[53]](#footnote-53)**

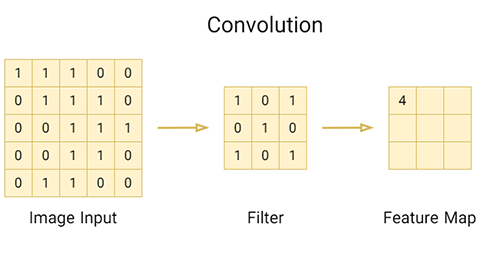
شبکه های عصبی کانولوشن ، دسته ای از شبکه های عصبی هستند که به طول معمول برای تحلیل تصاویر و گفتار در یادگیری ماشین و به طور جزئی تر در یادگیری عمیق استفاده می­شوند. این نوع شبکه دارای چندین لایه است که داده ها را پردازش کرده و ویژگی های مهم آنها را استخراج می­کنند. فیلترهای این شبکه بر اساس روش های اکتشافی[[54]](#footnote-54) بدست می آیند. شبکه های عصبی کانولوشن می‌توانند مهم‌ترین ویژگی فیلترها را بیاموزند و چون به پارامتر‌های زیادی احتیاج نیست، صرفه‌جویی زیادی در وقت و عملیات آزمون و خطا صورت می‌گیرد.

تا زمانی که با تصاویر با ابعاد بالا که هزاران پیکسل دارند کار نکنید، این صرفه‌جویی چندان به چشم نمی‌آید. هدف اصلی این الگوریتم ها این است که با حفظ ویژگی‌هایی که برای فهم آنچه داده‌ها نشان می‌دهند مهم هستند، داده‌ها را به فرم‌هایی که پردازش آن‌ها آسان‌تر است، درآورد.

شبکه های عصبی کانولوشن بر خلاف شبکه های عصبی معمولی ، به جای ضرب ماتریس از کانولوشن استفاده می­کند. این شبکه ها با اعمال فیلتر روی داده‌های ورودی شما پردازش خود را انجام می­دهند. چیزی که آن‌ها را بسیار خاص می‌کند، این است که شبکه های عصبی کانولوشن می‌توانند فیلترها را هم‌زمان با فرایند آموزش، تنظیم کنند. بنابراین، حتی وقتی مجموعه داده‌های عظیمی مانند تصاویر داشته باشید، نتایج به‌خوبی و در‌لحظه دقیق‌تر می‌شوند.

از آنجا که می‌توان فیلترها را برای آموزش بهتر شبکه های عصبی کانولوشن به روز رسانی کرد، نیاز به فیلترهای دستی از بین می‌رود و این انعطاف‌پذیری بیشتری در تعداد و ارتباط فیلترهایی که بر روی مجموعه داده‌ها اعمال می‌شوند، به ما می‌دهد. با استفاده از این الگوریتم، می‌توانیم روی مسائل پیچیده‌تری مانند تشخیص چهره کار کنیم.

# **7-2 کانولوشن**

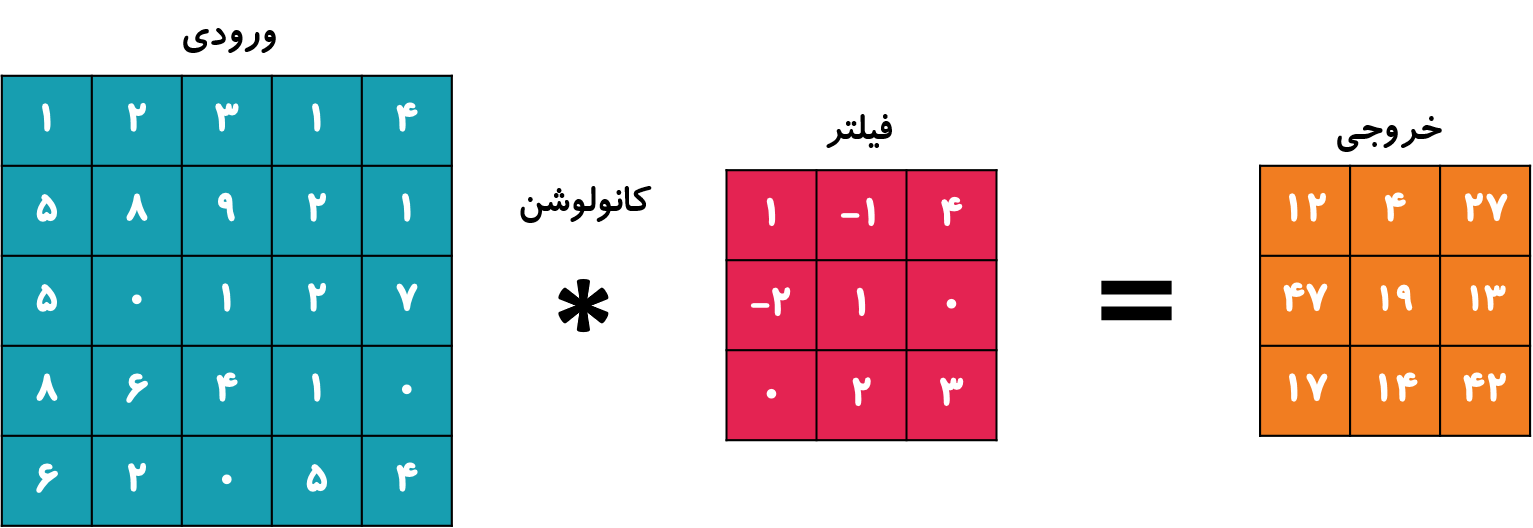


شکل 2-2 عمل کانولوشن

در عمل کانولوشن چند پارامتر اساسی وجود دارد که عبارتند از:

* ماتریس ورودی
* فیلتر کانولوشن
* عملگر کانولوشن
* ویژگی خروجی کانولوشن

کارکرد اصلی کانولوشن به این صورت است که عملگر کانولوشن، فیلتر کانولوشنی را برمی­دارد و روی ماتریس ورودی می­لغزاند.(فیلتر روی تصویر حرکت می­کند.) [8]



شکل 2-3 پارامتر های اصلی کانولوشن

اعدادی که در ماتریس خروجی ذخیره می‌شوند، تابعی از **ورودی و فیلتر** هستند. **فیلتر به دنبال پیدا کردن نواحی مشابه خود در تصویر است** و هرجایی ناحیه مشابهِ خود را پیدا کرد بلند فریاد می‌زند (عدد بزرگ).

**پس کانولوشن منجر به یافتن الگوهای خاص در تصویر باتوجه به فیلتر می‌شود.** اعداد موجود در فیلتر بسیار مهم هستند.

# **8-2 انواع شبکه های عصبی کانولوشن**

* **شبکه عصبی کانولوشن یک‌بعدی:**در این حالت، کرنل CNN در یک جهت حرکت می‌کند. CNNهای یک‌بعدی معمولاً روی داده‌های سری زمانی استفاده می‌شوند.
* **شبکه عصبی کانولوشن دو‌بعدی: در**این نوع از CNN، کرنل‌ها در دو جهت حرکت می‌کنند. CNN‌های دو‌بعدی در برچسب‌گذاری و پردازش تصویر کاربرد دارند.
* **شبکه عصبی کانولوشن سه‌بعدی:**این نوع CNN دارای کرنلی است که در سه جهت حرکت می‌کند. محققان از این نوع CNN در تصاویر سه‌بعدی مانند سی‌تی‌اسکن و MRI استفاده می‌کنند.

# **9-2 کاربرد های شبکه های عصبی کانولوشن**

* تشخیص دستخط های مختلف
* تشخیص تصاویر با پیش پردازش کم
* استفاده در سرویس های پستی برای خواندن کد پستی
* استفاده در بانکداری برای خواندن ارقام در چک
* کاربرد های بینایی کامپیوتر[[55]](#footnote-55)
* و ...

# **10-2 چرا از کانتینر های داکر در یادگیری عمیق استفاده می­کنیم؟**

اولین چیزی که قبل از صحبت در مورد کانتینرسازی باید درک کرد، مفهوم میکرو سرویس[[56]](#footnote-56) است. اگر یک برنامه بزرگ به سرویس های کوچکتر تقسیم شود، هر یک از آن سرویس ها یا فرآیندهای کوچک را می توان سرویس های میکرو نامید و از طریق یک شبکه با یکدیگر ارتباط برقرار می­کنند. رویکرد میکرو سرویس‌ ها برعکس رویکرد یکپارچه است که مقیاس‌بندی آن دشوار است. اگر یک ویژگی خاص دارای برخی مشکلات یا خرابی باشد، همه ویژگی ‌های دیگر مشابه آن را تجربه خواهند کرد. مثال دیگر این است که وقتی تقاضا برای یک ویژگی خاص به طور جدی در حال افزایش است، ما مجبور می­شویم منابعی مانند سخت افزار را نه تنها برای این ویژگی خاص بلکه برای کل برنامه افزایش دهیم و هزینه های اضافی را ایجاد کنیم که ضروری نیستند. این هزینه می تواند به حداقل برسد اگر رویکرد خدمات خرد با تقسیم برنامه به گروهی از خدمات کوچکتر در نظر گرفته شود. هر سرویس یا ویژگی های برنامه به گونه ای ایزوله شده است که می­توانیم بدون تأثیر بر سایر ویژگی های برنامه، مقیاس یا بروزرسانی کنیم. برای وارد کردن یادگیری ماشین به تولید، بیایید در نظر بگیریم که برنامه باید به سرویس‌های خرد کوچک‌تری مانند مصرف، آماده‌سازی، ترکیب، جداسازی، آموزش، ارزیابی، استنتاج، پس پردازش و نظارت تقسیم شود.[9]

## **1-10-2-کانتینرسازی**

معماری میکرو سرویس نیز دارای اشکالاتی است. هنگامی که برنامه یادگیری ماشین خود را در یک سرور توسعه می­دهید، به همان تعداد ماشین مجازی نیاز خواهید داشت که میکروسرویس حاوی وابستگی است. هر ماشین مجازی به یک سیستم‌عامل، کتابخانه‌ها و باینری‌ها نیاز دارد و منابع سخت‌افزاری بیشتری مانند پردازنده، حافظه و فضای دیسک را مصرف می‌کند، حتی اگر میکرو سرویس واقعاً در حال اجرا نباشد. به همین دلیل است که داکر وارد می­شود. اگر یک کانتینر در حال اجرا نباشد، منابع باقی مانده به منابع مشترک تبدیل می­شوند و برای سایر کانتینرها قابل دسترسی هستند. شما نیازی به اضافه کردن سیستم عامل در یک ظرف ندارید. بیایید یک راه حل کامل متشکل از برنامه های کاربردی 1 و 2 (مثلا APP 1 و APP 2) را در نظر بگیریم. اگر می‌خواهید APP 1 را کوچک کنید یا برنامه‌های کاربردی دیگری را اضافه کنید، می‌توانید با استفاده از منابع در دسترس از ماشین های مجازی به جای کانتینرها محدود شوید. اگر تصمیم دارید فقط APP1 و نه APP 2 را کوچک کنید (فقط یک مورد را نگه دارید)، APP 2 در حال تبدیل شدن به سهمی از همه فرآیندهای کانتینر است.

ایجاد یک مدل یادگیری ماشینی که در رایانه ما کار می کند واقعاً پیچیده نیست. اما وقتی برای مثال با مشتری کار می‌کنید که می‌خواهد از مدل در مقیاس استفاده کند، مدلی که می‌تواند در همه انواع سرورها در سراسر جهان کار کند، چالش‌برانگیزتر است. پس از توسعه مدل شما، ممکن است در لپ تاپ یا سرور شما به خوبی اجرا شود، اما واقعاً در سیستم های دیگر مانند زمانی که مدل را به مرحله تولید یا سرور دیگری منتقل می­کنید، کار نمی­کند. بسیاری از چیزها ممکن است رخ دهد مانند مشکلات عملکردی، از کار افتادن برنامه یا بهینه نبودن برنامه. موقعیت چالش برانگیز دیگر این است که مدل یادگیری ماشینی ما مطمئناً می تواند با یک زبان برنامه نویسی منفرد مانند پایتون نوشته شود، اما برنامه مطمئناً نیاز به تعامل با برنامه های کاربردی دیگر نوشته شده در زبان های برنامه نویسی دیگر برای جذب داده، آماده سازی داده، فرانت اند و غیره دارد. داکر اجازه می­دهد تا همه این تعاملات را بهتر مدیریت کنید زیرا هر میکرو سرویس را می توان به زبانی متفاوت نوشت که امکان مقیاس پذیری و افزودن یا حذف آسان سرویس های مستقل را فراهم می­کند. داکر تکرارپذیری، قابلیت حمل، استقرار آسان، بروزرسانی دانه ای، سبکی، سادگی را به ارمغان می­آورد.[9]

فصل دوم به پایان رسید. فصل بعدی پیرامون پیاده سازی پروژه ها به وسیله ابزار داکر است. در فصل بعدی ، چند پروژه یادگیری ماشین و یادگیری عمیق را با استفاده از ابزار داکر پیاده سازی می­کنیم و برای پروژه های خود ایمیج ساخته و از آن در پروژه خود استفاده می­کنیم.

# 

# فصل سوم

# پیاده سازی یادگیری ماشین/عمیق با استفاده از داکر

# **1-3 پروژه یادگیری ماشین**

# **1-1-3 مقدمات لازم برای ساخت ایمیج داکر**

ایمیج[[57]](#footnote-57) بر روی مجموعه ای از کد ها و متعلقات ساخته می­شود. بنابراین در ابتدا باید پروژه ای که قصد ساخت ایمیج آن را داریم ، انتخاب و فایل ها و اسناد مربوط به پروژه را جمع آوری کنیم.

نوار مغزی، الکتروانسفالوگرافی[[58]](#footnote-58) یا EEG  ثبت فعالیت الکتریکی مغز از طریق نصب الکترودهای سطحی بر روی سر و به صورت غیر تهاجمی می‌باشد. به طور کلی، در یک سیستم EEG، اثر الکتریکی فعالیت نورون‌های مغز از طریق الکترودهای نصب شده بر روی سر به دستگاه انتقال داده شده و پس از تقویت و حذف نویز به صورت سیگنال زمانی ثبت و نمایش داده می‌شود. سیگنال ثبت شده می‌تواند مستقیماً و یا پس از پردازش کامپیوتری توسط پزشک و یا متخصص علوم اعصاب مورد تحلیل قرار بگیرد[12].

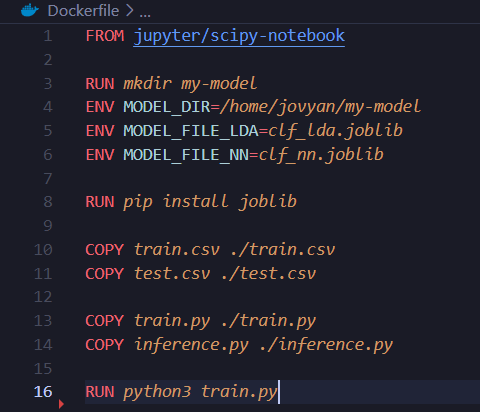
پروژه فرضی ما حاوی اسناد زیر می باشد:

* فایل train.py : یک فایل به زبان برنامه نویسی پایتون است که داده های EEG[[59]](#footnote-59) را از یک فایل به نام train.csv دریافت و نرمال­سازی[[60]](#footnote-60) می­کند. همچنین دو مدل را برای طبقه بندی داده ها آموزش می­دهد : تحلیل تشخیصی خطی و مدل چند لایه شبکه های عصبی.
* فایل inference.py : این فایل هم که به زبان برنامه نویسی پایتون نوشته شده ، برای استنتاج دسته ای با بارگذاری دو مدلی که در فایل قبلی ایجاد و آموزش داده شدند، فراخوانی می­شود. این برنامه داده های EEG جدید را که از یک دیتاست می آید ، نرمالسازی می کند. همچنین استنتاج روی مجموعه داده انجام داده و دقت طبقه بندی و پیش بینی ها را چاپ میکند.
* فایل train.csv : دیتاستی که توسط فایل train.py دریافت و آموزش داده می­شود.
* فایل test.csv : دیتاستی که توسط inference.py نرمالسازی می­شود.

# **2-1-3 شروع ساخت ایمیج**

حال ساخت ایمیج را آغاز می­کنیم.ساخت ایمیج بر روی Dockerfile صورت می­گیرد. برای ساخت ایمیج ابتدا یک فایل تحت نام Dockerfile در مسیر پروژه ایجاد کرده و آن را با یک ویرایشگر متن[[61]](#footnote-61) باز می­کنیم. حال دستورات لازم در Dockerfile نوشته می­شود تا ایمیج طبق این دستورات ساخته شود.

فرمت یک Dockerfile به شکل زیر است:



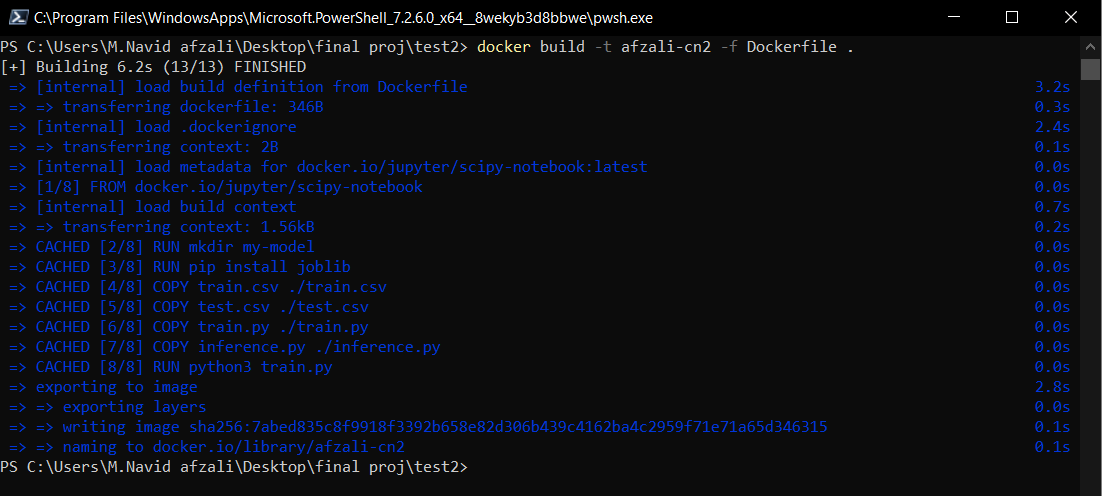
شکل 3-1 داکرفایل

* خط 1 : شروع ساخت ایمیج از یک ایمیج پایه : ما از ایمیج jupyter/scipy-notebook استفاده کرده­ایم. این ایمیج از مجموعه ای از ایمیج های آماده ی اجراست که شامل برنامه های کاربردی jupyter و ابزار های محاسباتی تعاملی می­باشد.
* خط 3 : ساخت یک دایرکتوری[[62]](#footnote-62) به نام my-model برای استفاده در ادامه کار
* خط 4-5-6 : برای آینده توسعه ی پروژه خود، می‌توان متغیرهای محیطی را از ابتدا، فقط یک بار در زمان ساخت، برای تداوم مدل آموزش‌دیده تنظیم کرد و شاید داده‌ها یا ابرداده‌های[[63]](#footnote-63) اضافی را به یک مکان خاص اضافه کرد. مزیت تنظیم متغیرهای محیطی این است که کار خود را بهتر با دیگران در یک ساختار دایرکتوری توافق شده به اشتراک بگذارید.
* خط 8 : اجرای دستور به منظور نصب پکیج joblib . Joblib مجموعه ای از ابزارها برای ارائه خط لوله[[64]](#footnote-64) سبک در پایتون است. Joblib به گونه ای بهینه سازی شده است که به ویژه در داده های بزرگ سریع و قوی باشد و دارای بهینه سازی های خاصی برای آرایه های numpy است. ما باید joblib را نصب کنیم تا سریال‌سازی[[65]](#footnote-65) و سریال‌زدایی[[66]](#footnote-66) مدل آموزش‌دیده‌مان را امکان‌پذیر کنیم.
* خطور 10 تا 14 : کپی کردن اسناد و فایل های پروژه و انتقال آن ها به ایمیج . این دستورالعمل برای کپی کردن فایل های جدید، از قسمت منابع [[67]](#footnote-67)به فایل سیستم هایی که در کانتینر قرار دارند است.
* خط 16 : اجرای دستور پایتون بر روی فایل train.py جهت نمایش خروجی هنگام اجرای ایمیج . ما train.py را اجرا می‌کنیم که مدل‌های یادگیری ماشین را به عنوان بخشی از فرآیند ساخت تصویر ما متناسب و سریال می‌کند.

در ادامه پس از نوشتن داکرفایل، وارد CMD یا ترمینال سیستم عامل خود شده و در مسیر فایل های پروژه ، با دستور زیر شروع به ساخت ایمیج از پروژه یادگیری ماشین خود می­کنیم:

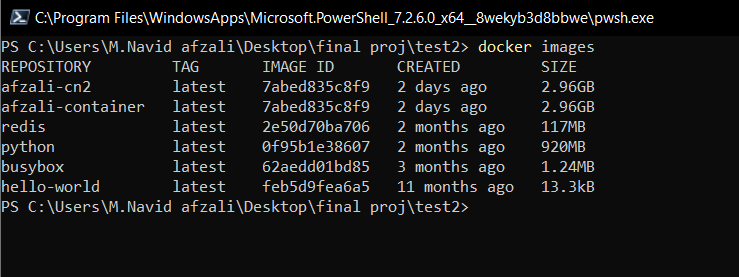


با اجرای این دستور، داکر شروع به ساخت ایمیج برای مدل یادگیری ماشین می­کند و با توجه به دستورات داکر فایل، ایمیج پایه و متعلقات آن را در صورت نیاز از مخزن داکر[[68]](#footnote-68) دریافت می­کند. خروجی نهایی به شکل زیر است:



شکل 3-2 مراحل ساخت ایمیج

حال ایمیج ما ساخته شده و در مخزن سیستم عامل میزبان موجود است:

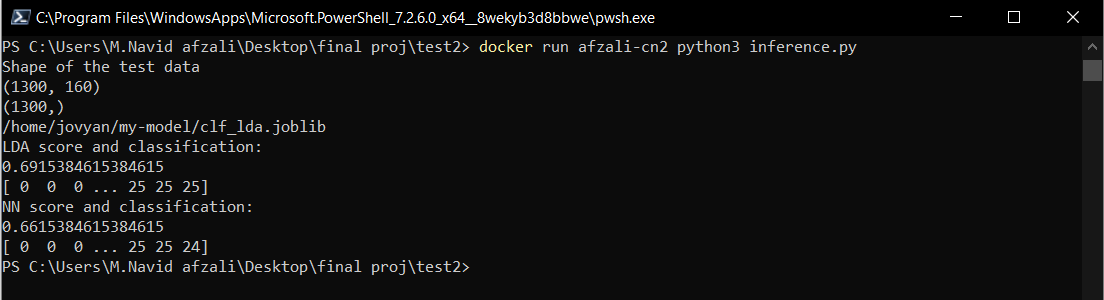


شکل 3-3 ایمیج های موجود در مخزن محلی (دریافت شده از مخزن Docker hub )

# 

# **3-1-3 اجرای ایمیج روی نگهدارنده**

حال می­توانیم استنتاج[[69]](#footnote-69) خود از مدل ها را روی داده های جدید یعنی داده های موجود در test.csv اجرا کنیم. به صورت زیر:



شکل 3-4 اجرای ایمیج روی نگهدارنده

همانطور که مشاهده می­شود ایمیج ساخته شده روی یک نگهدارنده اجرا شده و خروجی کد inference.py که طبقه بندی[[70]](#footnote-70) و دقت مدل ها را نمایش می­دهد، به نمایش در آمده است . میبینیم که مدل ها از دقت نسبتا خوبی برخوردارند. همچنین مسیر اجرای ایمیج را هم مشاهده می­کنید.

در ادامه یک پروژه یادگیری عمیق را مورد بررسی قرار می­دهیم و ایمیج مربوط به برنامه را می­سازیم.

# **3-2 پروژه یادگیری عمیق**

## **1-2-3 مقدمه**

در این پروژه یک مدل CNN که تصاویر مربوط به غذا ها را دسته­بندی می­کند را در قالب یک سرویس API پیاده سازی می­کنیم.

همانطور که در فصل دوم شرح داده شد، **شبکه عصبی کانولوشن** نوع خاصی از شبکه عصبی با چندین لایه است که داده‌هایی را که آرایش شبکه‌ای دارند، پردازش کرده و سپس ویژگی‌های مهم آن‌ها را استخراج می‌کند. یک مزیت بزرگ استفاده از CNNها این است که نیازی به انجام پیش‌پردازش زیادی روی تصاویر نیست.

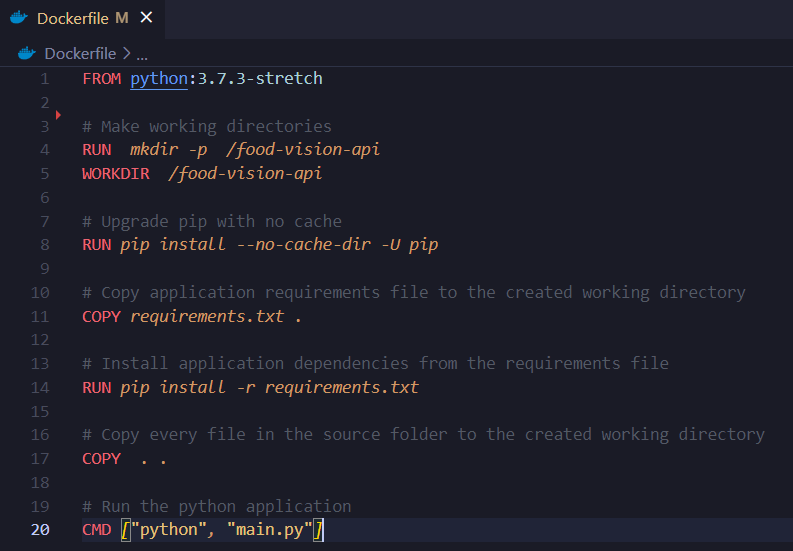
FastAPI یک فریمورک تحت وب پایتون است و به وسیله آن برنامه های api را با سرعت بالا ایجاد می­کنند.

پروژه حاوی اسناد زیر است:

* Main.py : فایل اصلی پروژه به زبان پایتون که یادگیری روی مدل را انجام می­دهد ؛ در نهایت روی یک پورت لوکال سرویس fastAPI را اجرا می­کند.
* Requirments.txt : یک فایل متنی است که در آن پکیج های مورد نیاز پروژه نوشته شده است. از آن در dockerfile هنگام دریافت پکیج ها به وسیله دستور pip استفاده می­شود.
* Dockerfile : یک فایل داکر که در آن نحوه ساخت ایمیج نوشته شده است و ایمیج طبق این فایل ساخته می­شود.

## **2-2-3 شروع ساخت ایمیج**

محتویات Dockerfile به شکل زیر است(تصویر در صفحه بعد):



شکل 3-5 محتویات dockerfile

* خط 1: ایمیج پایه را نام می­بریم؛ در این پروژه ما از ایمیج پایتون نسخه 3.7 استفاده کرده ایم.
* خط4و5: مانند پروژه قبل ساخت دایرکتوری برای ادامه کار را انجام می­دهد.
* خط 8: یک دستور پایتون است که پکیج pip را بروزرسانی می­کند.
* خط 11: فایل متنی پروژه یعنی requirments.txt را کپی می­کند و در دایکتوری ای که در خطوط قبلی ساختیم قرار می­دهد.(دایرکتوری در کانتینر ساخته می­شود.)
* خط 14: دستور پایتون است که با دستور pip محتویات فایل requirments.txt را دریافت می­کند.
* خط 17: تمام فایل های پروژه را کپی می­کند و در دایرکتوری ساخته شده قرار می­دهد.
* خط 20: یک دستور اجرایی است که فایل main.py را اجرا می­کند؛ معادل دستور زیر در CMD است:

Python main.py

در ادامه پس از نوشتن داکرفایل، وارد CMD یا ترمینال سیستم عامل خود شده و در مسیر فایل های پروژه ، با دستور زیر شروع به ساخت ایمیج از پروژه خود می­کنیم:

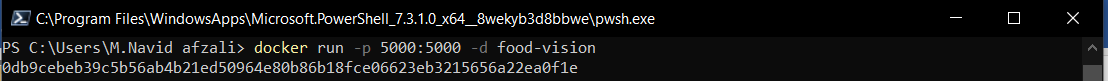
docker build -t <your-image-name> .

نحوه ساخت همانند شکل 16 است.

پس از ساخت ایمیج با دستور زیر آن را در یک کانتینر اجرا می­کنیم:

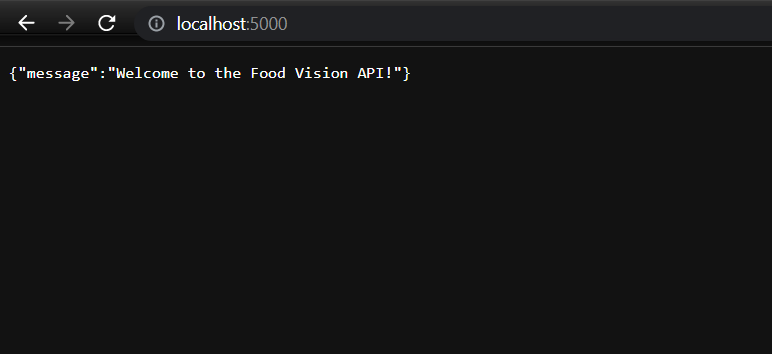
docker run -p 5000:5000 -d <your-image-name>

با تنظیمات -p پورت 5000 کامپیوتر خود را به پورت 5000 کانتینر متصل می­کنیم؛ همچنین با تنظیمات -d کانتینر را به صورت detached mode اجرا می­کنیم؛ به این معنی که در پس زمینه اجرا می­شود. در صورت موفقیت آمیز بودن اجرا ، شناسه کانینر در ترمینال برمیگردد.



شکل 3-6 اجرای ایمیج2

اکنون با باز کردن صفحه <http://localhost:5000/> در رایانه خود می­توانیم مشاهده کنیم که آیا کانتینر با موفقیت اجرا شده است یا خیر:



شکل 3-7 اجرای برنامه با کانتینر

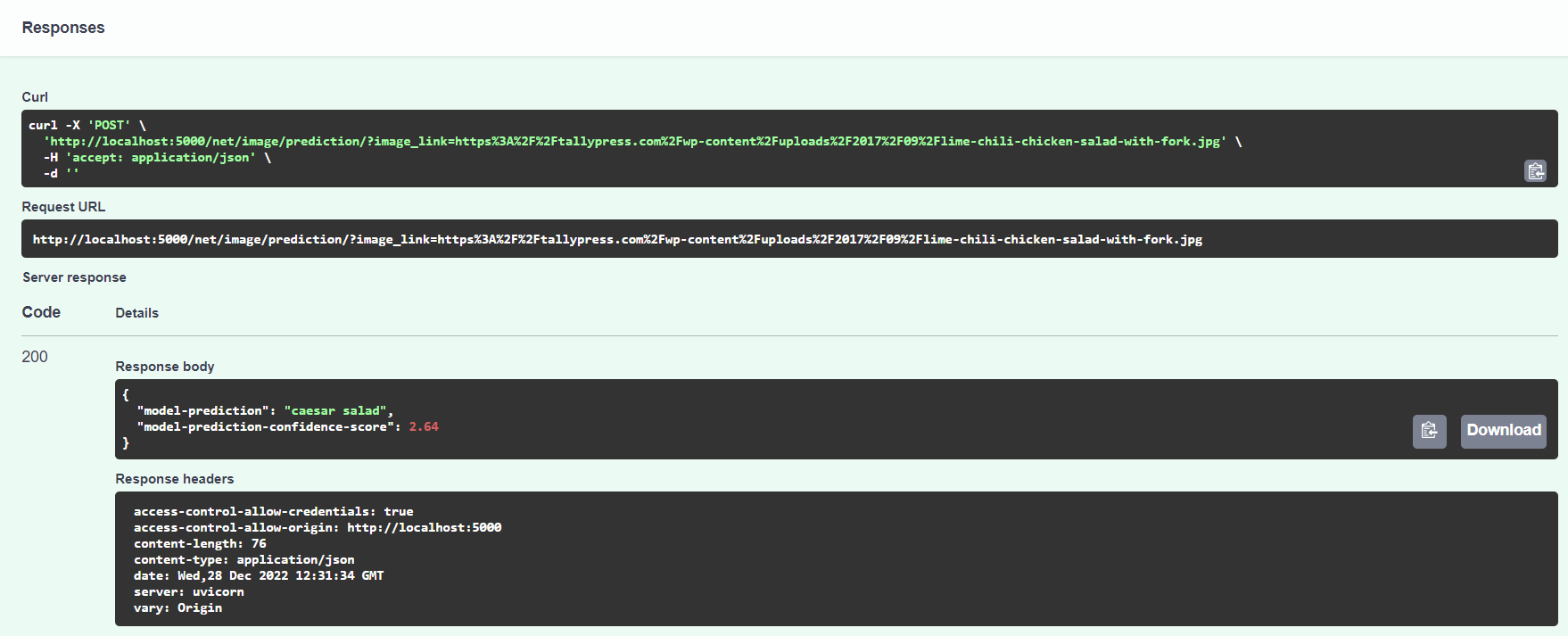
با وارد کردن دستور <http://localhost:5000/docs> به صفحه fastAPI هدایت می­شویم؛ در آن جا می­توانیم با متد POST یک تصویر ارسال کنیم و نتیجه طبقه بندی تصویر را در قالب JSON تماشا کنیم.

برای مثال ما از تصویر زیر استفاده می کنیم:



شکل 3-8 تصویر نمونه

خروجی fastAPI به صورت زیر می­باشد:



شکل 3-9 خروجی کانتینر تحت وب

# **3-3 جمع بندی و نتیجه گیری**

همانطور که در بخش های قبل مشاهده کردید، ساختن ایمیج برای یک پروژه یادگیری ماشین و یادگیری عمیق با نوشتن داکرفایل برای آن و اجرای دستور docker build … امکان پذیر است.

حال ما با push کردن ایمیج خود در مخزن Docker hub می­توانیم پروژه یادگیری ماشین خود را در دسترس عموم قرار دهیم تا با دریافت آن ایمیج و اجرای آن روی یک نگهدارنده در سرور میزبان خود، بتوانند بدون آشنایی با کتابخانه ها و متعلقات آن پروژه ، پروژه را اجرا و نتیجه آن را مشاهده نمایند. فرضا پروژه با زبان python3 نوشته شده و فرد دریافت کننده ایمیج روی سیستم خود python2 را دارد. با داکر دیگر مشکلی برای اجرا وجود نخواهد داشت و دریافت کننده نیازی به تغییر نسخه python خود نیز ندارد؛ همچنین به تبع رفع این مشکل ، فرد نیازی به استفاده از ماشین مجازی نیز ندارد و در نتیجه سرعت و فضای سیستم را از دست نخواهد داد.

در این پروژه دیدیم که علاوه بر سایر برنامه ها ، برای برنامه های یادگیری ماشین نیز می­توان ایمیج داکر ساخت و از آن استفاده کرد. . این تنها یک مثال از ساخت ایمیج و اجرای آن برای پروژه یادگیری ماشین است. برای سایر برنامه های یادگیری ماشین نیز به همین ترتیب عمل می­شود.

استفاده از ایمیج داکر برای پروژه های یادگیری ماشین، باعث راحتی، سرعت و اتلاف وقت بسیار کمتر توسعه دهندگان در استفاده از آن در سایر سرور ها و سیستم های خارجی است.

# **سخن پایانی**

در پایان تشکر می­کنم از استاد محترم دکتر راستگو که بنده را در نوشتن این پایان نامه کمک کردند و باعث شدند که بنده با این ابزار کاربردی و خوب و همچنین با مفاهیم جالب یادگیری عمیق آشنا شوم. موفقیت های روز افزون را برای ایشان از ایزد منان خواهانم.

و تشکر میکنم از همه شما خوانندگان گرامی بابت صبر و شکیبایی که در خواندن این پایان نامه از خود نشان دادید.

همگی موفق و پیروز باشید.

محمدنوید افضلی.

# **منابع**

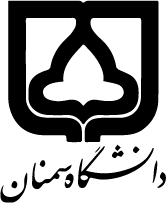
1. <https://docs.docker.com>
2. **کتاب "لذت آموزش گام به گام یادگیری عمیق با مثال عددی، کتاب دوم" اثر دکتر کیانی و دکتر راستگو**
3. <https://docker.com/>
4. <https://dockerme.ir/>
5. <https://youtube.com/>
6. <https://blog.faradars.org/%D8%A2%D9%85%D9%88%D8%B2%D8%B4-%D8%AF%D8%A7%DA%A9%D8%B1-docker-%D8%B1%D8%A7%DB%8C%DA%AF%D8%A7%D9%86/>
7. <https://quera.org/blog/what-is-a-convolutional-neural-network/>
8. <https://howsam.org/convolutional-neural-network/>
9. <https://towardsdatascience.com/why-using-docker-for-machine-learning-74c927ceb6c4>
10. <https://virgool.io/DockerMe/%DA%AF%D8%B1%DB%8C%D8%B2-%D8%A7%D8%B2-%D8%AA%D8%AD%D8%B1%DB%8C%D9%85-%D8%AF%D8%A7%DA%A9%D8%B1-%D8%A8%D8%A7-%DA%86%D9%86%D8%AF-%D8%B1%D9%88%D8%B4-z6czoxibqnyk>
11. https://wikipedia.org
12. https://negand.com/fa/35-2/

# **Abstract**

Nowadays, “Docker” is mentioned as one of the important and practical tools in the field of programming. Docker as a tool for building and running programs in container environments has significantly increased the speed and productivity of these programs. With Docker, we will no longer worry about the difference between the versions of the program's components and its libraries.

### Docker is a set of platform as a service products that use OS-level virtualization to deliver software in packages called containers. Docker takes away repetitive, mundane configuration tasks and is used throughout the development lifecycle for fast, easy and portable application development – desktop and cloud. Currently, this tool can be used in the development of programs related to artificial intelligence and machine(deep) learning, and developers in this field can share their programs with public or their colleagues without worry through Docker. So we won’t have troubles like “your app is not working in my local.” and so on.[3]

### Also Docker is so helpful for these projects because machine(deep) learning projects have many libraries with several versions. On the other hand , they’re not same in most devices, so in these cases Docker help us! In this project, we are going to build and implement a container (explained in the introduction) on our machine(deep) learning model using the tools available in Docker. In fact, the idea of this project is to build a quick and easy Docker container with a simple machine(deep) learning model and run it.

****

**Faculty of Electrical and Computer Engineering**

**Department of Computer Engineering**

**Bachelor project thesis**

Title:

**Using Docker to implement deep learning projects**

*Supervisor:*

**Dr.Rastgoo**

Student:

**Mohammad navid afzali**

Student number:

**9711126012**

**first semester of 01-02**

1. Container [↑](#footnote-ref-1)
2. Deep learning [↑](#footnote-ref-2)
3. Docker [↑](#footnote-ref-3)
4. Open source [↑](#footnote-ref-4)
5. build [↑](#footnote-ref-5)
6. run [↑](#footnote-ref-6)
7. Process container [↑](#footnote-ref-7)
8. Linux container [↑](#footnote-ref-8)
9. Red Hat [↑](#footnote-ref-9)
10. Linux [↑](#footnote-ref-10)
11. Open-shift [↑](#footnote-ref-11)
12. Fedora [↑](#footnote-ref-12)
13. Docker client [↑](#footnote-ref-13)
14. Commercial [↑](#footnote-ref-14)
15. Container [↑](#footnote-ref-15)
16. Virtual machine [↑](#footnote-ref-16)
17. Kernel [↑](#footnote-ref-17)
18. Python زبان برنامه نویسی [↑](#footnote-ref-18)
19. Hypervisor-Based [↑](#footnote-ref-19)
20. Workload [↑](#footnote-ref-20)
21. real time [↑](#footnote-ref-21)
22. Docker engine [↑](#footnote-ref-22)
23. Docker client [↑](#footnote-ref-23)
24. User interface [↑](#footnote-ref-24)
25. User friendly [↑](#footnote-ref-25)
26. Docker deamon [↑](#footnote-ref-26)
27. Docker registry [↑](#footnote-ref-27)
28. Docker image [↑](#footnote-ref-28)
29. Libraries [↑](#footnote-ref-29)
30. Template [↑](#footnote-ref-30)
31. Docker hub [↑](#footnote-ref-31)
32. Repository [↑](#footnote-ref-32)
33. **cloud -hosted** [↑](#footnote-ref-33)
34. Cgroups [↑](#footnote-ref-34)
35. CPU [↑](#footnote-ref-35)
36. I/O disk [↑](#footnote-ref-36)
37. Docker engine [↑](#footnote-ref-37)
38. Namespaces [↑](#footnote-ref-38)
39. Mount [↑](#footnote-ref-39)
40. Command line [↑](#footnote-ref-40)
41. Options [↑](#footnote-ref-41)
42. Running [↑](#footnote-ref-42)
43. Images ID [↑](#footnote-ref-43)
44. Tag [↑](#footnote-ref-44)
45. login [↑](#footnote-ref-45)
46. Dockerfile [↑](#footnote-ref-46)
47. Base image [↑](#footnote-ref-47)
48. Cache [↑](#footnote-ref-48)
49. این آزمون در مقاله معروف تورینگ **دستگاه محاسباتی و هوشمندی** مطرح شد و هدف این تست بررسی هوشمندی ماشین بود. در این آزمون شخص C به شکلی غیر مستقیم با شخص B و یک ماشین (A) در ارتباط است و نمی‌داند کدامشان ماشین است. او از آنها سوالاتی می‌پرسد و اگر از روی جواب‌ها نتواند شخص B را از ماشین تشخیص دهد، می‌گوییم که ماشین تست تورینگ را قبول شده است. [↑](#footnote-ref-49)
50. Machine learning [↑](#footnote-ref-50)
51. Neural Network [↑](#footnote-ref-51)
52. node [↑](#footnote-ref-52)
53. Convolutional Neural Network (CNN) [↑](#footnote-ref-53)
54. Heuristic [↑](#footnote-ref-54)
55. Computer vision [↑](#footnote-ref-55)
56. Micro-service [↑](#footnote-ref-56)
57. Image [↑](#footnote-ref-57)
58. Electroencephalography [↑](#footnote-ref-58)
59. EEG نوسانات ولتاژ ناشی از جریان یونی در نورون های مغز را اندازه گیری می­کند. از نظر بالینی، EEG به ثبت فعالیت الکتریکی خود به خودی مغز در طی یک دوره زمانی اشاره دارد، همانطور که از چندین الکترود قرار داده شده بر روی پوست سر ثبت می­شود.(ویکی پدیا) [↑](#footnote-ref-59)
60. Normalize [↑](#footnote-ref-60)
61. Text editor [↑](#footnote-ref-61)
62. Directory-Repository [↑](#footnote-ref-62)
63. Meta data [↑](#footnote-ref-63)
64. Pipeline [↑](#footnote-ref-64)
65. Serialization [↑](#footnote-ref-65)
66. Deserializaiton [↑](#footnote-ref-66)
67. Source [↑](#footnote-ref-67)
68. Docker Hub [↑](#footnote-ref-68)
69. Inference [↑](#footnote-ref-69)
70. Classification [↑](#footnote-ref-70)