دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده مهندسی برق و کامپیوتر

گزارش مینی‌پروژه اول درس شبکه عصبی

یلدا فروتن

۸۱۰۱۹۶۲۶۵

استاد درس

جناب آقای دکتر کلهر

پاییز ۹۸

۱. مرور مسائل پیرامون CNN

۱,۱ انتخاب توابع و روش‌های بهینه‌سازی

۱,۱,۱ مقایسه توابع فعال‌ساز

ازآنجایی‌که مسائل مطرح‌شده در شبکه‌های عصبی از نوع Logistic Regression نیست، عموماً در لایه‌های مخفی از توابع غیرخطی استفاده می‌شود و توابع خطی پاسخگو نخواهد بود. دلیل این امر آن است که استفاده از تابع خطی در لایه‌های پنهان بی‌فایده است؛ زیرا در هنگام Update کردن پارامترها، می‌توان بدون استفاده از توابع خطی و با اعمال ضرایبی به وزن‌ها، توابع خطی را پیاده‌سازی کرد. تنها استفاده از توابع فعال‌ساز خطی در شبکه‌های عصبی، در لایه آخر مسائل مربوط به رگرسیون است. حتی در این نوع مسائل نیز، در لایه‌های پنهان از توابع غیرخطی همچون ReLU، Tanh و Sigmoid استفاده می‌شود.

برای اعمال روش‌های بهینه‌سازی همچون روش Gradient Descent لازم است تابع فعال‌ساز مشتق‌پذیر باشد. بنابراین تابع در یک بازه خاص نباید دارای مقدار 0 باشد. در ادامه سه تابع فعال‌ساز غیرخطی ReLU، Tanh و Sigmoid با یکدیگر مقایسه می‌شوند.

- **تابع ReLU**

تابع ReLU یا Rectified Linear Unit یک تابع ساده و کار راه‌انداز است. این تابع شبیه Ramp عمل می‌کند اما دیگر خطی نیست. همچنین همان‌طور که در ادامه آمده است تابع ReLU در مقادیر مثبت همچون تابع همانی است اما برای مقادیر منفی دارای مقدار صفر است.

$$ReLU: a = \max(0, z)$$

$$Leaky\ ReLU: a = \max(0.01z, z)$$

مزایا

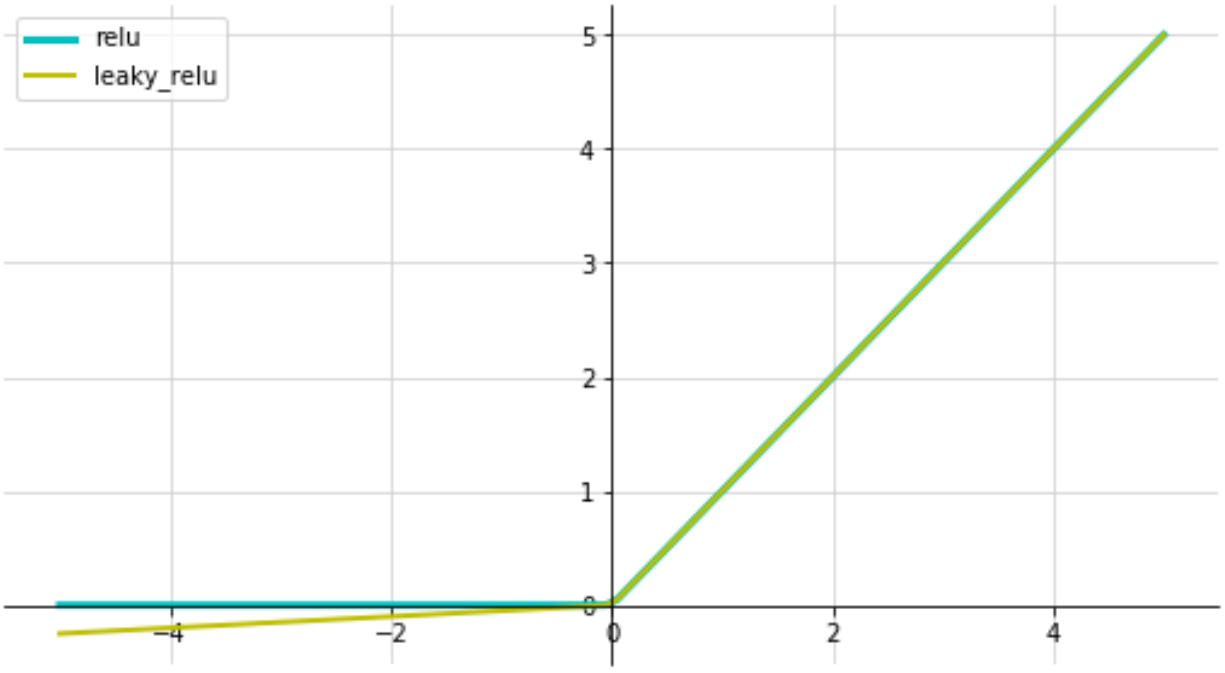
- مشتق تابع ReLU به ازای مقادیر مثبت، غیر صفر و یک‌به‌یک است؛ بنابراین مشکل Gradient Vanishing، که در قسمت بعد تعریف می‌شود، ندارد.
- تابع ReLU نسبت به سایر توابع فعال‌ساز ساده‌تر است.

معایب

- در هنگام استفاده از روش‌های Gradient Search، به ازای مقادیر منفی، مشتق تابع ReLU صفر خواهد شد و مقادیر وزن‌ها update نخواهد شد. راه‌حل آن قرار دادن مشتق تقریبی برای مقادیر منفی است که به این تابع جدید Leaky ReLU می‌گویند.
- فقط به‌عنوان تابع فعال‌ساز در لایه‌های میانی شبکه می‌توان از آن استفاده کرد.
- تابع ReLU غیرخطی است اما درواقع از دو بخش خطی ساخته شده است.

کاربرد

- تابع ReLU را می‌توان به‌عنوان Defalt Function نام‌گذاری کرد؛ در مواردی که رفتار تابع مشخص نیست، می‌توان از ReLU یا Leaky ReLU استفاده کرد. به‌طورکلی تابع ReLU یک General Approximator است. تابع فعال‌ساز Leaky ReLU بهتر از تابع ReLU است اما کاربرد آن کمتر است.



شکل ۱ – توابع فعال‌ساز ReLU و Leaky ReLU

- **تابع Sigmoid**

تابع Sigmoid شبیه به تابع Sign نرم‌شده است که مقادیر مختلف ورودی را به بازه ۰ تا ۱ می‌برد.

$$Sigmoid: a = \frac{1}{1 + e^{-z}}$$

مزایا

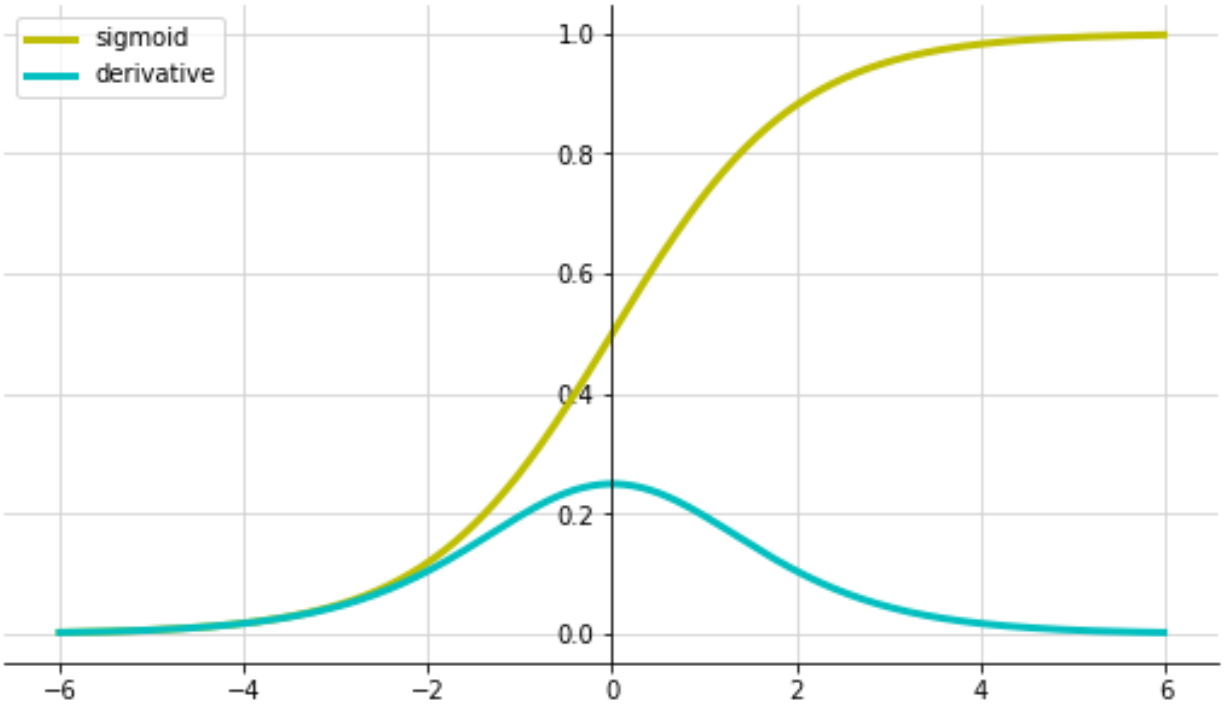
- شاید مهم‌ترین فایده تابع Sigmoid آن است که تابع پله را به‌صورت غیرخطی پیاده‌سازی می‌کند. همچنین در بازه -۳ تا ۳، تغییرات کوچک در ورودی تابع، منجر به تغییرات بزرگ در خروجی می‌گردد.
- تابع Sigmoid و حتی بخش‌هایی از آن همچنان غیرخطی است.
- تابع Sigmoid و مشتق آن Smooth هستند.

معایب

- یکی از معایب تابع Sigmiod آن است که تابع بین ۰ تا ۱ بوده و نسبت به ۰ متقارن نیست؛ بنابراین تمام مقادیر آن مثبت هستند.
- مشتق تابع فعال‌ساز Sigmoid همواره بین ۰ تا ۱ است. ازآنجایی‌که در روش‌های کاهش خطا به‌صورت Gradient Search، مشتقات در زنجیره‌های طولانی در یکدیگر ضرب می‌شوند، احتمال کوچک کردن (نزدیک به صفر شدن) وزن‌های مربوط به لایه‌های ابتدایی شبکه زیاد است که به آن Gradient Vanishing می‌گویند زیرا اثر گرادیان در لایه‌های عقب‌تر را کم می‌کند و مؤلفه خطای بازگشتی منجر به یادگیری نمی‌شود.

کاربرد

- تابع Sigmiod برای کاربردهای Classifation مناسب است و سریع‌تر از ReLU فرایند یادگیری را همگرا می‌سازد.
- تابع Sigmoid برای لایه خروجی مسائل Binary Classification که دارای دو کلاس هستند، مناسب است.



شکل ۲ – تابع فعال‌ساز Sigmoid و مشتق آن

- **تابع Tanh**

تابع Tanh درواقع شیفت‌یافته تابع Sigmoid است که به‌صورت زیر تعریف می‌شود.

$$Tanh: a = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

مزایا

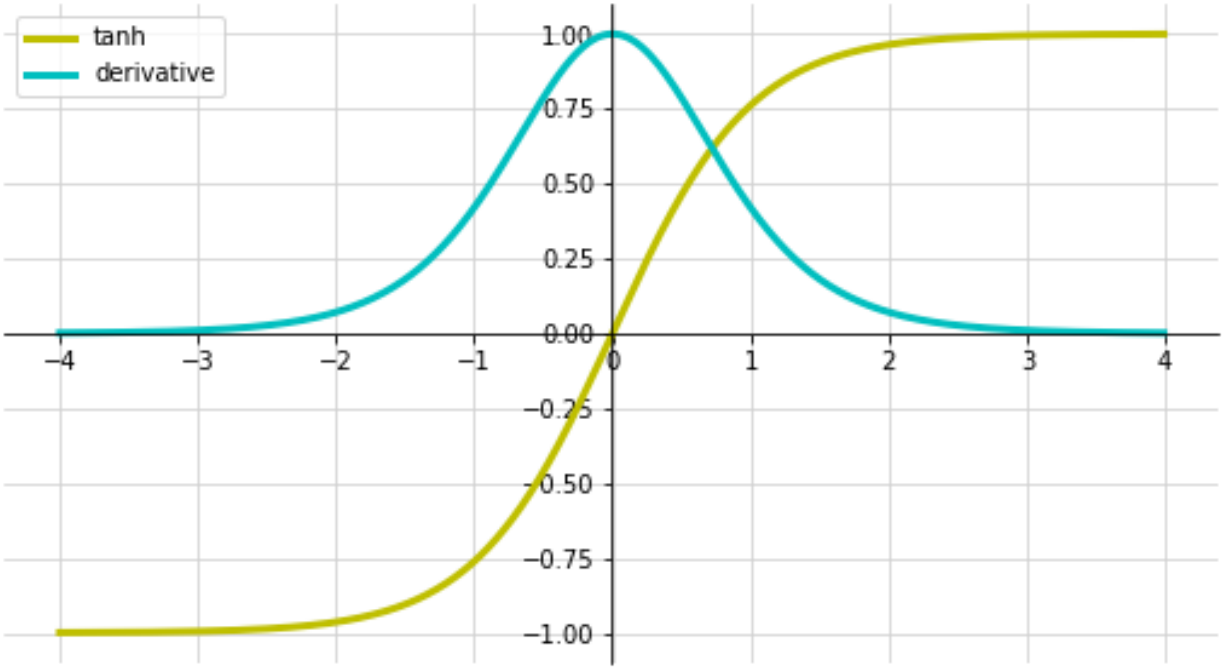
- تابع Tanh عملکردی شبکه به Sigmoid دارد با این تفاوت که نسبت به صفر متقارن است. بنابراین فرایند یادگیری برای آن آسان‌تر است.
- گرادیان تابع Tanh نسبت به گرادیان تابع Sigmoid، بزرگ‌تر بوده که فرایند بهینه‌سازی را سریع‌تر می‌کند.

معایب

- تابع Tanh همانند Sigmoid منجر به Gradient Vanishing می‌شود.

کاربرد

- در شبکه‌های عصبی به‌عنوان تابع فعال‌ساز لایه‌های پنهان مناسب است اما برای لایه‌های خروجی مناسب نیست.



شکل ۳ – تابع فعال‌ساز Tanh و مشتق آن

بنابراین تابع ReLU به دلیل داشتن مقادیر متفاوت از صفر برای مقادیر مثبت، نسبت به دو تابع Sigmoid و Tanh ارجحیت دارد.

۲،۱،۱ توابع Loss

توابع Loss Function در راستای آموزش شبکه عصبی مفید هستند؛ به‌گونه‌ای که با داشتن ورودی‌ها و خروجی‌های ایده‌آل (Target)، می‌توان میزان خطا بین خروجی شبکه (Predicted) و Target را محاسبه کرد. توابع خطا را می‌توان براساس نوع مسئله شبکه عصبی دسته‌بندی کرد.

- **مسائل Regression**: در این مسائل خروجی Target به‌صورت پیوسته است و شبکه عصبی یک تابع را تقریب می‌زند. تابع خطا رایج برای این‌گونه مسائل تابع Mean Square Error یا MSE است.

- **مسائل Classification**: مقادیر خروجی در مسائل Classification به‌صورت گسسته بوده و نمایانگر Label ورودی‌ها است. تابع خطا مرسوم برای این نوع مسائل، Cross-entropy است.
- **مسائل Embedding**: در این نمونه مسائل، مقایسه‌ای بین دو مقدار ورودی صورت می‌گیرد که آیا شبیه به هم هستند یا نه. تابع خطا رایج برای این مسائل، Hinge Loss است.

در ادامه چهار نمونه تابع خطا رایج به همراه مزایا و کاربرد آن‌ها بررسی شده است.

- **Hinge Loss**

یکی از جایگزین‌های تابع Cross-entropy برای مسائل Binary Classification تابع خطا Hinge است که برای مدل‌های طبقه‌بندی در ماشین‌های بردار پشتیبانی (SVMs) کاربرد دارد. درواقع تابع Hinge برای مسائل Binary Classification هنگامی‌که Target دارای مقادیر ۱) است، به کار گرفته می‌شود تا نمونه‌ها را به سمت داشتن علامت مناسب مثبت یا منفی بکشاند. در برخی موارد استفاده از تابع خطا Hinge باعث بهبود کارایی شبکه نسبت به استفاده از خطا Cross-entropy در مسائل Binary Classification شده است.

همان‌طور که گفته شد، لازم است خروجی مربوط به Target دارای مقادیر -۱ یا +۱ باشد. همچنین خروجی پیش‌بینی شده باید مقادیری در بازه [-1,1] دارا باشد. بنابراین استفاده از تابع فعال‌ساز Tanh مناسب است زیرا خروجی آن در بازه مدنظر قرار دارد.

$$Hinge\ Loss: J(y) = \sum_{i=1}^{n} \max(0,1 - t_i.h_i)$$

در رابطه بالا منظور از t، خروجی Target و منظور از h خروجی پیش‌بینی‌شده است.

- **Softmax Cross Entropy**

منظور از Softmax Cross Entropy، تابع فعال‌ساز Softmax و تابع هزینه Cross entropy است. درواقع تابع Softmax، یک حالت Soft از تابع ماکزیمم است که احتمال ماکزیمم و نزدیک به آن را می‌دهد. ورودی تابع Softmax یک بردار N بعدی و خروجی آن یک بردار بین ۰ تا ۱ است که در ادامه آمده است.

$$P_i = \frac{e^{a_i}}{\sum_{k=1}^{N} e_k^a}$$

ازآنجایی‌که تابع Softmax یک توزیع احتمالی به‌عنوان خروجی می‌دهد، در لایه خروجی شبکه‌های عصبی از آن استفاده می‌گردد. در هنگام Backpropagation لازم است مشتق و یا گرادیان Softmax محاسبه گردد که در ادامه آمده است.

$$\frac{\partial p_i}{\partial a_j} = \frac{\partial \frac{e^{a_i}}{\sum_{k=1}^{N} e_k^a}}{\partial a_j} = \cdots = p_i(\delta_{ij} - p_j)$$

تابع هزینه Cross Entropy فاصله بین توزیع واقعی مدل و توزیعی که شبکه به دست آورده را مشخص می‌کند که در ادامه رابطه آن آمده است.

$$H(y,p) = -\sum_i y_i \log(p_i)$$

درنهایت از مشتق تابع فعال‌ساز Softmax برای محاسبه مشتق تابع خطا Cross entropy استفاده می‌شود.

- **Mean Squared Error**

مجموع مربعات خطا، تابع پیش‌فرض مسائل Regression است. این خطا به‌صورت میانگین مجذور اختلاف بین خروجی‌های پیش‌بینی شده و خروجی Target محاسبه می‌گردد. MSE جدای از علامت Predict و Target، همواره دارای مقدار مثبت است و بهترین مقدار خطا برای آن صفر خواهد بود. استفاده از مجذور خطا نمایانگر آن است که اشتباهات بزرگ‌تر منجر به خطاهای بیش‌تر می‌شود.

$$MSE\ Loss: J(y) = \frac{1}{n}\sum_{i=1}^{n}(t_i - h_i)^2$$

- **Log Loss**

تابع Log Loss یا Binary Cross Entropy برای مسائل Classification کاربرد دارد. برای نمونه اگر خروجی‌های مسئله به‌صورت دو کلاس باشند، از تابع Bi nary Cross Ent ropy استفاده می‌شود و رابطه آن به‌صورت زیر است:

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i . \log\big(p(y_i)\big) + (1-y_i)\,.\log(1-p(y_i))$$

به‌عنوان‌مثال اگر دو کلاس وجود داشته باشد و برچسب‌های صفر و یک را به آن‌ها اختصاص داده شده باشد، تابع loss به‌صورت زیر محاسبه می‌گردد:

$$y = ۰ \rightarrow H_p(q) = {}^{-۱}\!/_{N}\left(\log\left(۱ - p(y)\right)\right) \rightarrow H_p(q) = \begin{cases} ۰ \\ \infty \end{cases}$$

$$y = ۱ \rightarrow H_p(q) = {}^{-۱}\!/_{N}\log\left(۱ - p(y)\right) \rightarrow H_p(q) = \begin{cases} ۰ \\ \infty \end{cases}$$

بنابراین اگر target مربوط به یک کلاس ۰ باشد و سیستم به‌درستی ۰ پیش‌بینی کرده باشد، در این صورت مقدار خطا صفر خواهد بود. اما اگر شبکه به‌اشتباه ۱ تشخیص دهد، مقدار خطا بین‌هایت می‌شود. همین شیوه برای سایر label ها نیز در نظر گرفته شده است.

۳,۱,۱ روش‌های بهینه‌سازی

- **Gradient Descent**

گرادیان کاهشی مهم‌ترین تکنیک برای آموزش شبکه‌های عصبی و روشی برای بهینه‌سازی سیستم‌های هوشمند است. گرادیان کاهشی عمدتاً در راستای آپدیت کردن وزن‌های شبکه برای مینیمم کردن تابع خطا استفاده می‌شود. می‌دانیم که شبکه عصبی با تکنیک‌های Backpropagation آموزش داده می‌شود؛ بدین‌صورت که شبکه ابتدا در مسیر forward، ورودی‌ها و وزن‌های متناظر با آن‌ها ضرب داخلی کرده و با یک بایاس جمع و سپس تابع فعال‌ساز را بر روی آن پیاده می‌کند. سپس در مسیر Backward، شبکه با استفاده از اپتیمایزر مقادیر وزن‌ها و بایاس‌ها را آپدیت می‌کند. در روش گرادیان کاهشی، آپدیت کردن پارامترها با کمک گرادیان تابع خطا در مسیر برگشت است. روابط مربوط به گرادیان کاهشی در ادامه آمده است. لازم به ذکر است تابع خطا مستقل از اپتیمایزر بوده و همانند قسمت قبل باید انتخاب گردد.

$$w = w - \alpha \frac{\partial}{w\partial} J(w)$$

$$b = b - \alpha \frac{\partial}{b\partial} J(w)$$

علاوه بر انتخاب تابع هزینه، باید مقدار)که نشان‌گر نرخ یادگیری است تعیین گردد. اگر)کوچک انتخاب شود، همگرایی تضمین شده اما زمان آموزش شبکه طولانی می‌گردد. درحالی‌که با انتخاب)بزرگ، شبکه سریع‌تر یاد می‌گیرد اما ممکن است واگرا گردد. درواقع انتخاب نرخ یادگیری، الگوریتم مشخصی ندارد.

مدل استاندارد گرادیان کاهشی گرادیان همه داده‌ها را به‌صورت یکجا محاسبه می‌کند و همه داده‌ها را فقط یک‌بار آپدیت می‌کند. این در حالتی است روش گرادیان برای دیتاست‌های بزرگ کند و غیرقابل‌کنترل می‌شود. بنابراین از روش Stochastic Gradient Descent برای حل مشکل گفته‌شده استفاده می‌شود. روش SGD برای هر نمونه از داده‌های Training، عمل آپدیت کردن پارامترها را انجام می‌دهد که منجر به‌سرعت بیشتر نسبت به GD می‌شود. بنابراین روابط گرادیان برای حالت Stochastic به‌صورت زیر می‌گردد. x(i) و y(i) بیان‌گر نمونه‌های Training set هستند.

$$\theta = \theta - \alpha \frac{\partial}{\theta \partial} J(\theta; x(i); y(i))$$

به دلیل به‌روزرسانی مکرر پارامترها در روش SGD، تابع خطا نوسان خواهد کرد. بنابراین احتمال یافتن مینیمم‌های محلی و بهتر نسبت به روش GD افزایش می‌یابد. از طرفی به‌روزرسانی‌های مکرر، همگرایی به یک نقطه مینیمم محلی را سخت‌تر کرده و منجر به Overshooting تابع هزینه می‌گردد. البته با کاهش نرخ یادگیری می‌توان همگرایی را همانند روش GD به دست آورد.

مشکل عدم ثبات همگرایی در روش SGD منجر به استفاده از روش Mini-Batch Gradient Descent می‌شود.

درواقع روش Mini-Batch GD مزیت هر دو روش GD و SGD را دارد که به ازای هر n نمونه از مجموعه Training، پارامترهای شبکه را آپدیت می‌کند. این روش یادگیری را به سمت همگرایی با ثبات بیشتر می‌برد.

روش گرادیان و حتی تغییرات آن (SGD و Mini-Batch GD) همچنان دارای مشکلاتی هستند. به‌عنوان‌مثال انتخاب Learning Rate مناسب همچنان مسئله است. همچنین همه پارامترها با نرخ یادگیری یکسان آپدیت می‌شوند. درحالی‌که برای پترن‌های کم‌تکرار نرخ یادگیری بزرگ‌تر مطلوب است. علاوه‌برآن توابع هزینه به دلیل non-convex بودن، دارای چندین نقطه مینیمم هستند. بدین‌صورت یافتن بهترین نقطه (Global Optimum و داشتن Domain of Attraction) سخت خواهد بود چراکه معمولاً تابع در یک نقطه مینیمم

محلی گیر می‌کند. همچنین وجود نقاط Saddle Point مشکل‌ساز خواهد بود؛ درواقع این نقاط در یک بعد مینیمم بوده و در بعد دیگر ماکزیمم هستند.

- **Momentum**

در روش SGD به دلیل افزایش واریانس، همگرایی مشکل می‌شود. استفاده از روش Momentum منجر به افزایش سرعت SGD می‌گردد؛ بدین‌صورت که Momentum منجر به کاهش تغییرات عمودی (مسیر نامربوط) و افزایش تغییرات افقی (مسیر مربوط) می‌شود. درواقع Momentum علاوه بر گرادیان قدم کنونی، کسری از گرادیان قدم‌های پیشین را برای آپدیت کردن بردار کنونی استفاده می‌کند.

$$V_{dw} = \beta V_{dw} + (1-\beta)dw$$

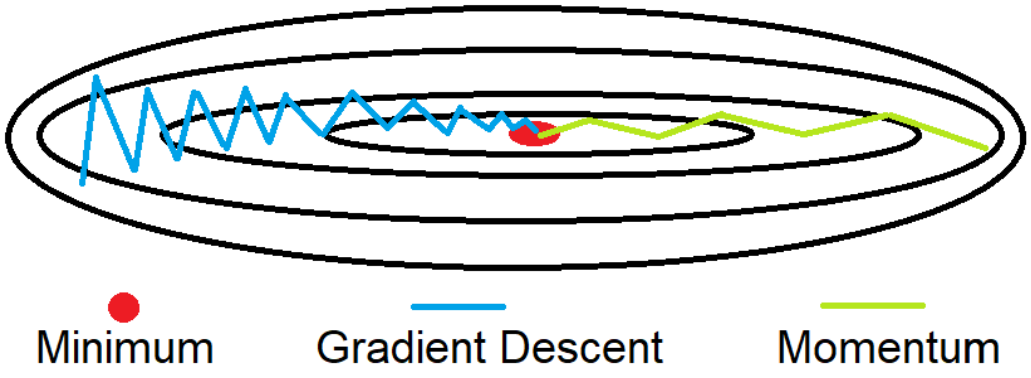
$$V_{db} = \beta V_{db} + (1-\beta)db$$

رایج‌ترین مقدار برای پارامتر β، ۰,۹ است. بدین معنا که برای آپدیت کردن پارامترهای مدنظر از ۹ گرادیان قبلی میانگین می‌گیرد. درنهایت پارامترها به‌صورت زیر آپدیت می‌شوند.

$$w = w - \alpha V_{dw}$$

$$b = b - \alpha V_{db}$$

درنتیجه هنگام آپدیت شدن پارامترها با کمک momentum، همگرایی با سرعت بیشتر اتفاق می‌افتد و نوسانات را کاهش می‌دهد.



شکل ۴ – مقایسه همگرایی تابع خطا در روش Momentum و Gradient Descent

- **AdaDelta**

درواقع بهینه‌سازی تابع خطا به روش‌های Adaptive راه‌حلی برای انتخاب Learning Rate براساس پارامترهای شبکه هستند. به‌گونه‌ای‌که برای پارامترهای پرتغییر، آپدیت‌های کوچک و برای پارامترهای که تغییرات اندک دارند از آپدیت‌های بزرگ استفاده می‌کنند. بنابراین برای هر پارامتری براساس گرادیان قبلیِ، از نرخ یادگیری متفاوتی استفاده می‌شود. بنابراین در روش Adagerad از نرخ یادگیری متفاوت برای هر پارامتر)در هر قدم t استفاده می‌شود. در رابطه زیر، منظور از $g_{t,i}$ گرادیان تابع خطا است. بدیهی است که مزیت روش‌های Adaptive آن است که لازم نیست نرخ یادگیری به روش دستی تنظیم شود.

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\alpha}{\sqrt{G_{t,ii} + \varepsilon}} \cdot g_{t,i}$$

در روش Adagrad، ازآنجایی‌که مجذور گرادیان‌های قبلی جمع می‌شوند، Learning Rate کوچک و کوچک‌تر می‌گردد. کاهش Learning Rate، منجر به کاهش سرعت یادگیری شبکه می‌شود. برای حل این مشکل، الگوریتم AdaDelta پیشنهاد شده است. در AdaDelta به جای ذخیره کردن مجذور همه گرادیان‌های قبلی، بخشی از آن‌ها را ذخیره می‌کنند. به‌طوری‌که به جای ذخیره مجذور گرادیان همه پارامترهای w قبلی، مجموع گرادیان‌های قبلی به شکل بازگشتی به‌عنوان میانگین مجذور گرادیان پارامترهای قبلی تعریف می‌گردد. حال میانگین کنونی در step=t به میانگین قبلی و گرادیان کنونی وابسته است که به‌صورت زیر تعریف می‌گردد. لازم به ذکر است)همان ترم momentume است که در حدود ۰,۹ انتخاب می‌شود.

$$E[g^2](t) = \beta E[g^2](t-1) + (1-\beta) g^2(t)$$

بنابراین در روش AdaDelta علاوه بر محاسبه Learning Rate برای هر پارامتر، از Momentum نیز استفاده می‌شود. همچنین از مشکل Vabishing که در Adagrad مطرح شد.

- **Adam**

در روش AdaDelta از نرخ یادگیری متفاوت برای آپدیت کردن هر پارامتر استفاده می‌شود. در روش Adam(Adaptive Moment Estimation) علاوه بر Learning Rate، از ترم Momentum متفاوت نیز استفاده می‌شود.

$$\widehat{m_t} = \frac{m_t}{1 - \beta_1^t}$$

$$\widehat{v_t} = \frac{v_t}{1 - \beta_2^t}$$

بنابراین الگوریتم Adam از رابطه زیر برای آپدیت کردن پارامترها استفاده می‌کند.

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{\widehat{v_t}} + \varepsilon} . \widehat{m_t}$$

مقادیر رایج برای β_1، β_2 و)به ترتیب ۰,۹، ۰,۹۹۹ و ۱۰$^{-8}$ است. درواقع روش Adam منجر به همگرایی سریع‌تر نسبت به سایر الگوریتم‌های بهینه‌سازی می‌شود. همچنین با مشکلات سایر الگوریتم‌ها همچون همگرایی نرخ یادگیری به صفر و کاهش سرعت همگرایی تابع خطا، مواجه نمی‌شود.

به‌طورکلی الگوریتم Adam بهتر از سایر الگوریتم‌های Adaptive عمل می‌کند. حال اگر داده‌های ورودی به اصطلاح sparse باشند، روش‌هایی مانند SGD و Momentum ضعیف عمل می‌کنند و باید از روش‌های Adaptive استفاده کرد. برای دستیابی به همگرایی سریع‌تر در مدل‌های عمیق و پیچیده، الگوریتم Adam بهتر از سایرین عمل می‌کند.

۲,۱ تکنیک‌هایی برای بهبود CNN

- **تکنیک Batch Normalization**

دیده شده که نرمالایز کردن ورودی به دلیل هم‌مقیاس کردن، حساسیت یادگیری شبکه نسبت به مقیاس‌های متفاوت را کاهش می‌دهد. از طرفی هر Feature Map یک ورودی برای لایه‌های بعدی است. بنابراین بهتر است قبل از هر Hiddern Layer، داده‌ها نرمالایز شوند. درواقع Batch Normalization باعث می‌شود توزیع داده‌های Test و Train نزدیک به هم شوند. بدین‌صورت که در هر بعد، مقادیر مربوط به هر sample را با یکدیگر جمع کرده و میانگین و سپس واریانس هر بعد را محاسبه می‌کند. بنابراین داده‌های هر بعد منهای میانگین و تقسیم بر واریانس مربوط به بعد خود می‌شوند. البته از یک‌ترم)نیز استفاده شده است تا درصورت صفر شدن واریانس، داده‌ها به بین‌هایت میل نکنند. با این کار ورودی‌ها دارای میانگین صفر و واریانس یک خواهند شد. سپس با استفاده از پارامترهای)و)بر روی داده‌های جدید عمل Scale و Shift را پیاده می‌کند. پارامترهای)و)با استفاده از روش‌های Gradient Search، Learn می‌شوند. درواقع Batch Normalization سعی می‌کند

پراکندگی هر بعد را، مستقل از بعدهای دیگر، سفید بکند تا قدرت Partitioning را بهبود بخشد و با نحوه پیاده‌سازی Batch Normalization بر روی داده‌ها آمده است.

Input: Values of x over a mini-batch: $B = \{x_{1\ldots m}\}$;

Parameters to be learned: γ, β

Output: $\{y_i = BN_{\gamma,\beta}(x_i)\}$

$$\mu_B \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\sigma_B^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_B)^2$$

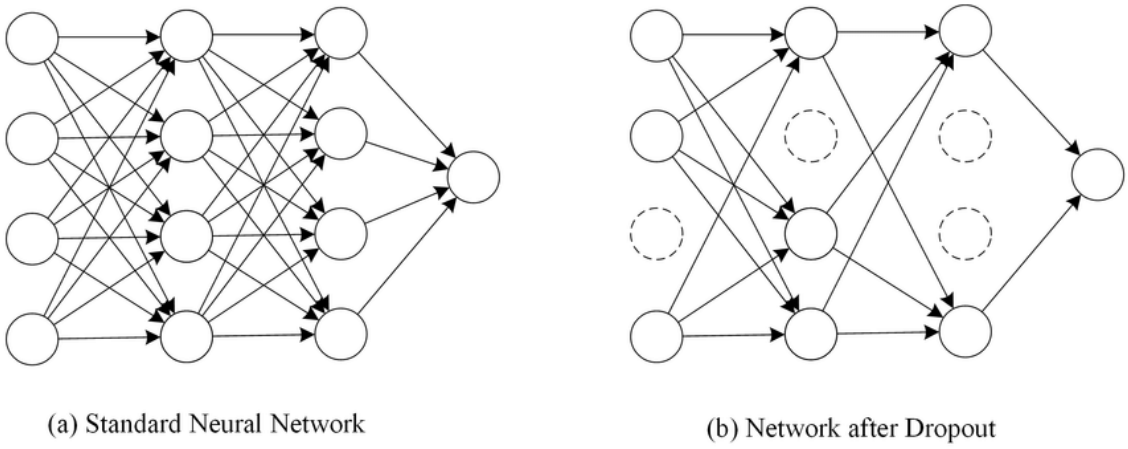
$$\widehat{x_i} \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}}$$

$$y_i \leftarrow \gamma \widehat{x_i} + \beta \equiv BN_{\gamma,\beta}(x_i)$$

- **تکنیک Drop Out**

تکنیک Drop Out، در هر ایپاک به‌صورت تصادفی درصدی از نورون‌های هر لایه را deactive می‌کند و سعی دارد با نورون‌های باقی‌مانده، شبکه را در مرحله call و recall آموزش دهد. در ایپاک بعد، ماسک تصادفی برای انتخاب نورون‌های فعال و غیرفعال عوض می‌شود اما درصد آن‌ها ثابت است. همانند Regularization term، هربار تابع به شکل تصادفی نادقیق‌تر می‌شود. البته اینکار منجر به کاهش دقت شبکه نمی‌شود؛ درواقع Drop Out با حذف برخی نورون‌ها، نویز تصادفی به شبکه اضافه می‌کند تا شبکه از Domain of Attraction های بد خارج شود و حتی با partially کار کردن برخی نورون‌ها، شبکه از Domain of Attraction خوب خارج نشود. البته زمانی می‌توان برخی نورون‌ها را غیرفعال کرد که به شبکه redundancy افزوده شده باشد تا با استفاده از Drop Out فضای پارامتری با دقت یادگیری و قدرت تعمیم بهتر ایجاد شود. لازم به ذکر است با خاموش کردن برخی نورون‌ها لازم است وزن نورون‌های فعال افزایش یابد تا قدرت فلو اطلاعاتی کاهش نیابد. بنابراین Drop Out قدرت یادگیری شبکه را بهتر کرده و شبکه را نسبت به خاموش شدن برخی نورون‌ها مقاوم می‌سازد. همچنین پس از پایان روند یادگیری، شبکه از قدرت تعمیم بهتری برخوردار خواهد بود. عملکرد Drop

Out در شکل زیر نشان داده شده است. شکل سمت چپ، شبکه را به‌صورت Fully Connected نشان می‌دهد. پس از اعمال Drop Out شبکه به‌صورت Partially Wireing شده در شکل سمت راست قابل‌مشاهده است.



شکل ۵ - اعمال تکنیک Drop Out و نحوه سیم‌بندی

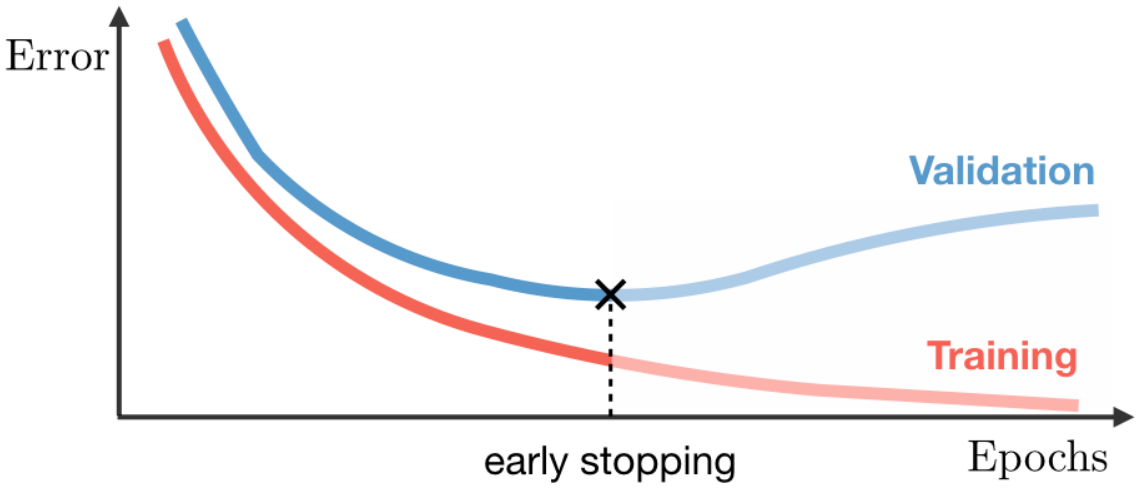
درواقع Drop Out یک Hyperparameter است و برای اعمال آن لازم است برخی نکات در نظر گرفته شود که در ادامه آمده است:

- برای لایه ورودی نباید از Drop Out استفاده کرد یا حداقل ۹۰ درصد ورودی باید به شبکه اعمال گردد.
- ممکن است یک لایه دارای ابعاد بزرگی نسبت به سایر لایه‌ها باشد. در این حالت شبکه نسبت به لایه مذکور بیشترین احتمال Overfitting را دارد. با استفاده از Drop Out می‌توان یک سری نورون‌های مربوط به آن لایه را به‌صورت رندوم حذف کرد. البته در لایه‌های دیگر نیز می‌توان از Drop Out استفاده کرد اما لایه‌ای که احتمال Overfitting در آن بیش‌تر است، نورون‌های بیش‌تری در آن حذف می‌شود.
- در ابتدا لازم است بدون Drop Out تابع Loss محاسبه گردد و اگر تابع به‌صورت Monotonicly Decreasing بود، Drop Out بر روی لایه‌های مدنظر اعمال گردد.
- برای داده‌های Test نباید از تکنیک Drop Out استفاده کرد.

- **تکنیک Early Stopping**

در نمودار Loss Function برحسب تعداد Epoch، اگر خطای مربوط به داده‌های Training رسم شود، مشاهده می‌شود که به‌صورت Monotonicly Decreasing است. حال این سؤال مطرح می‌شود که تا کجا فرایند Learning شبکه و Update کردن پارامترها ادامه پیدا کند؟ و کدام Loss مناسب خواهد بود؟ در این قسمت از داده‌های Validation کمک گرفته می‌شود. بدین‌صورت که نمودار خطا مربوط به داده‌های

Validation دارای یک نقطه مینیمم است و پس‌ازآن نقطه خطا داده‌های Validation افزایش یافته و شبکه قدرت تعمیم خود را از دست می‌دهد و به‌اصطلاح Overfit می‌شود. واضح است که در این نقطه تابع J یا همان خطای مربوط به داده‌های Training به‌طور کامل کاهش نیافته است و به آن Early Stopping می‌گویند. یکی از فواید داده‌های Validation آن است که در توقف فرایند یادگیری شبکه کمک می‌کنند.



شکل ۶ – نمودار خطا داده‌های Training و Validation برحسب تعداد ایپاک

- **تکنیک Norm Penalty**

ممکن است یک شبکه عصبی در فرایند یادگیری از دقت مناسب برخوردار باشد اما برای داده‌های Validation یا Test درست عمل نکند و به‌اصطلاح Overfit شود. به‌عنوان‌مثال اگر یک شبکه دارای Dataset کوچک باشد، معمولاً با مشکل Overfitting همراه است. بدین‌صورت که شبکه همه ویژگی‌های داده‌های Training حتی اطلاعات مربوط به نویز و پترن‌های کم‌تکرار را حفظ می‌کند. درواقع شبکه داده‌های Training را یاد نگرفته بلکه در حافظه خود ذخیره کرده است. یکی از راه‌حل‌های افزایش قدرت تعمیم شبکه برای داده‌های جدید، استفاده از Regularization است. به‌گونه‌ای که در Loss Function، یک Regularization Term نیز با تابع جمع می‌شود. تابع خطا بدون در نظر گرفتن Regularization، تنها خروجی را به Target نزدیک می‌کرد؛ اما با استفاده از Regularization Term، سعی می‌شود تا اندازه وزن‌ها نیز کوچک بماند و جواب‌هایی که در کرانه‌های فضا قرار دارند، در نظر گرفته نشوند.

رویکرد Regularization براساس محدود کردن گنجایش مدل‌های شبکه عصبی است که با افزودن Norm Penalty به تابع خطا ممکن می‌شوند. ترم)یک Hyperparemeter است که میزان استفاده از

Regularization را نشان می‌دهد. بدیهی است که $\alpha=0$ هیچ‌ترم رگولاریزیشنی به تابع خطا اضافه نمی‌کند. لازم به ذکر است بایاس‌ها درترم رگولاریزیشن دخالت داده نمی‌شوند.

$$J(\theta; X, y) = J(\theta; X, y) + \alpha\Omega(\theta)$$

حال Norm Penalty می‌تواند انواع مختلفی داشته باشد. استفاده از L1 Norm مجموع وزن‌های شبکه را با تابع خطا جمع می‌کند. رایج‌ترین شیوه پیاده‌سازی Regularization استفاده از L2 Norm است که مجموع توان دوم وزن‌ها را با تابع خطا جمع می‌کند.

۲. طبقه‌بندی دیتاست Fashion MNIST

هدف از این بخش، طبقه‌بندی داده‌ها مربوط به Fashin MNIST با استفاده از Convolutional Neural Network و مقایسه شبکه طراحی‌شده با شبکه‌های کلاسیک است. دیتاست Fashion MNIST، دارای ۷۰۰۰۰ تصویر Grayscale در ابعاد ۲۸) ۲۸ است. همانند دیتاست MNIST، ۶۰۰۰۰ نمونه به‌عنوان داده‌های Training و ۱۰۰۰۰ نمونه باقی‌مانده به‌عنوان داده‌های Test به کار گرفته می‌شوند. خروجی‌های شبکه، ده دسته بوده که شامل ده نمونه لباس است. در ادامه ۲۵ نمونه از داده‌های Fashion MNIST به همراه برچسب مربوطه آمده است.

شکل ۷ – ۲۵ نمونه از داده‌های Fashion MNIST

۱,۲ طراحی یک شبکه CNN

شبکه عصبی مدنظر برای طبقه‌بندی داده‌های ورودی دیتاست با استفاده از Framework کراس طراحی شده است. دیتاست Fashion MNIST با استفاده از Keras در برنامه فراخوانی شده است. سپس داده‌های ورودی و خروجی نرمالایز شده‌اند؛ پیش‌تر گفته شد که ابعاد تصاویر ۲۸) ۲۸ پیکسل شامل مقادیر ۰ تا ۲۵۵ هستند. بنابراین به منظور مستقل از بعد شدن شبکه، پیکسل تصاویر Training و Test به عدد ۲۵۵ تقسیم شده‌اند. حال شبکه مدنظر به‌صورت زیر طراحی گردید.

- **لایه Conv2D اول**

داده‌های Training به‌عنوان ورودی شبکه به یک لایه Conv شامل ۶۴ فیلتر با ابعاد ۳) ۳ داده شده است. بدین‌صورت که یک Patch ۳) ۳ در تصاویر ورودی sweep شده و خانه‌های متناظر patch و فیلتر ضرب داخلی شده و سپس با یکدیگر و با بایاس جمع می‌شوند. درنهایت تابع فعال‌ساز ReLU بر آن‌ها پیاده شده و یک Feature Map ساخته می‌شود. به دلیل استفاده از ۶۴ فیلتر، ۶۴ Feature Mage ساخته می‌شود. از آنجا که ابعاد ورودی و فیلتر به ترتیب ۲۸) ۲۸ و ۳) ۳ بوده، ابعاد Feature Map های ساخته شده، ۲۶) ۲۶ خواهد شد. بنابراین در این لایه از Padding استفاده نشده و Stride=1 در نظر گرفته شده است. لازم به ذکر است مقادیر فیلتر اعم از وزن‌ها و بایاس در هنگام استفاده از Optimizer در راستای مینیمم‌سازی تابع خطا مشخص می‌شوند. این پارامترها برای هر فیلتر شامل ۹ وزن و یک بایاس بوده که برای ۶۴ فیلتر، ۶۴۰ پارامتر می‌شوند.

- **لایه Maxpooling اول**

در لایه قبل ۶۴ Feature Map با ابعاد ۲۶) ۲۶ ساخته شد. حال با استفاده از یک لایه Maxpooling عمل Scaling بر روی ورودی‌های جدید پیاده می‌شود. اندازه Maxpooling ۲*۲ است که مقدار ماکزیمم ۴ خانه مجاور را انتخاب کرده و به Feature Map انتقال می‌دهد. عمل Maxpooling در اینجا منجر به نصف شدن ابعاد تصاویر می‌گردد. بنابراین Feature Map های جدید دارای ابعاد ۱۳*۱۳ خواهند بود. بدیهی است که Maxpooling تغییری در تعداد ورودی‌های به وجود نمی‌آورد و همچنین پارامتری برای Learn شدن ندارد.

- **لایه Conv2D دوم**

درواقع خروجی Maxpooling به‌صورت ورودی این لایه است؛ پس ورودی این لایه به‌صورت ۶۴ Feature Map با ابعاد ۱۳*۱۳ خواهد بود. در این قسمت نیز مجدد از ۶۴ فیلتر ۳*۳ استفاده می‌شود و تابع فعال‌ساز ReLU بر آن‌ها اعمال می‌گردد. بنابراین خروجی این لایه ۶۴ Feature Map با ابعاد ۱۱*۱۱ خواهد بود.

- **لایه MaxPooling دوم**

در این بخش نیز از یک لایه MaxPooling با ابعاد ۲*۲ استفاده شده است که ماکزیمم مقدار ۴ پیکسل کنار هم را به Feature Map انتقال می‌دهد. لازم به ذکر است ازآنجایی‌که ابعاد ورودی این بخش ۱۱*۱۱ بوده و ابعاد Maxpooling ۲*۲ است، در هر ردیف خانه‌های آخر در Maxpooling شرکت داده نمی‌شوند. بنابراین خروجی به‌صورت ۶۴ Feature Map با ابعاد ۵*۵ خواهد بود. بدیهی است در این بخش همانند Maxpooling اول، پارامتری برای Learn شدن وجود ندارد.

- **لایه Flatten**

از لحاظ هندسی، شبکه جداپذیر شده است و می‌توان همانند شبکه‌های MLP با آن برخورد کرد. بنابراین با افزودن دو لایه Dense می‌توان طبقه‌بندی را تکمیل نمود. حال با استفاده از لایه Flatten، ویژگی‌های استخراج‌شده در لایه قبل به فرم یک بردار درمی‌آیند. خروجی لایه قبل، ۶۴ Feature Map با ابعاد ۵*۵ بود. بنابراین به‌صورت ۱۶۰۰ درایه برای یک بردار هستند. درواقع ویژگی‌های باقیمانده شامل Frequent Pattern ها بوده و nullity و disturtion از آن‌ها حذف گردیده است.

- **لایه Dense اول**

در این لایه ۱۶۰۰ ویژگی استخراج‌شده به ۱۲۸ نورون به‌صورت Fully Connected داده می‌شود. بدیهی است که ورودی در وزن مربوطه ضرب داخلی و با بایاس جمع می‌شود و سپس تابع ReLU بر آن‌ها پیاده می‌گردد. بنابراین تعداد پارامترها ۲۰۴۹۲۸ خواهد بود که شامل ۱۲۸+۲۰۴۸۰۰ است.

- **لایه Dense دوم**

در این لایه Feature های قبلی، ۲۰۴۹۲۸، با اعمال تابع فعال‌ساز Softmax به ۱۰ نورون داده می‌شود تا ۱۰ کلاس مدنظر خروجی شبکه گردند.

درنهایت شبکه طراحی‌شده به‌صورت زیر معرفی می‌گردد. به‌طورکلی الگوریتم Optimization باید ۲۴۳۷۸۶ پارامتر را برای شبکه آموزش دهد. Optimizer استفاده‌شده Adam است و از تابع خطا Sparse Categorical Cross-entropy استفاده شده است.

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_7 (Conv2D)            (None, 26, 26, 64)        640
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 13, 13, 64)        0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 11, 11, 64)        36928
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
flatten_4 (Flatten)          (None, 1600)              0
_________________________________________________________________
dense_7 (Dense)              (None, 128)               204928
_________________________________________________________________
dense_8 (Dense)              (None, 10)                1290
=================================================================
Total params: 243,786
Trainable params: 243,786
Non-trainable params: 0
```

شکل ۸ – خلاصه‌ای از شبکه تعریف شده

در نهایت با استفاده از Batch-size=32 و تعداد ۵۰ ایپاک، شبکه آموزش داده شد و به دقت ۹۹,۴۳ درصد برای داده‌های Training رسید. زمان آموزش شبکه در حدود ۳۶۸ ثانیه طول کشید. همچنین شبکه برای داده‌های Test به دقت ۹۰,۸۶ درصد رسید.

۲,۲ مقایسه شبکه طراحی شده با شبکه‌های عصبی ساده

شبکه Multi Layer Perceptron با تک‌لایه مخفی توانایی تقریب توابع convex را با دقت کافی دارد. درحالی که برای مسائل non-convex باید تعداد نورون‌های شبکه تک‌لایه را زیاد کرد. روش بهتر افزودن یک لایه مخفی دیگر است. حال شبکه MLP با دو لایه مخفی، می‌توان توابع non-convex را تقریب زد. اگر ورودی‌ها corrolation، distortion و disturbance نداشته باشند، دولایه مخفی کفایت می‌کند و افزودن لایه‌های دیگر redundancy ایجاد می‌کند.

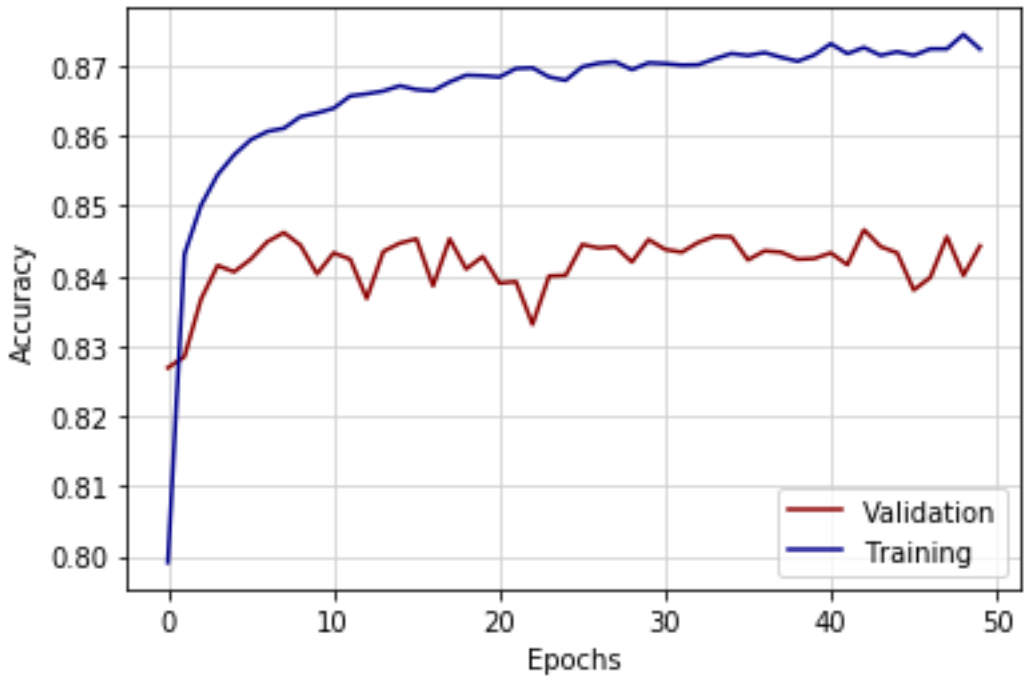
در این قسمت شبکه CNN طراحی‌شده با سه شبکه عصبی ساده با تعداد ۰، ۱ و ۲ لایه مخفی مقایسه گردید. درابتدا معماری هر شبکه توضیح داده می‌شود و سپس چهار شبکه مذکور مقایسه می‌گردند.

منظور از صفر لایه مخفی، آن است که ورودی‌های شبکه به خروجی داده شوند. بنابراین از همان لایه خروجی شبکه CNN استفاده شده است با این تفاوت که ورودی‌ها نیز به آن داده شده است.

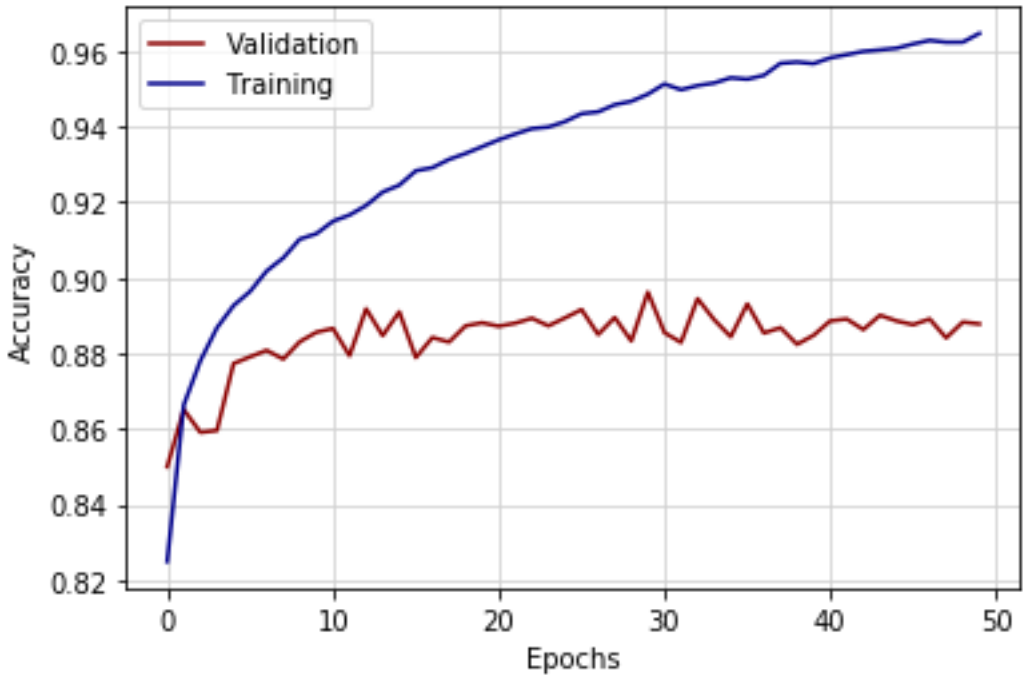
برای طراحی شبکه عصبی با یک لایه مخفی، ورودی‌ها به یک لایه با ۱۲۸ نورون داده شده و از تابع فعال‌ساز ReLU استفاده شده است. در نهایت خروجی لایه مخفی به یک لایه خروجی با ۱۰ نورون و تابع Softmax داده شده است.

در شبکه عصبی با دولایه مخفی، دو لایه Fully Connected با تعداد نورون‌های ۲۵۶ و ۳۲ و با تابع ReLU طراحی شده است.

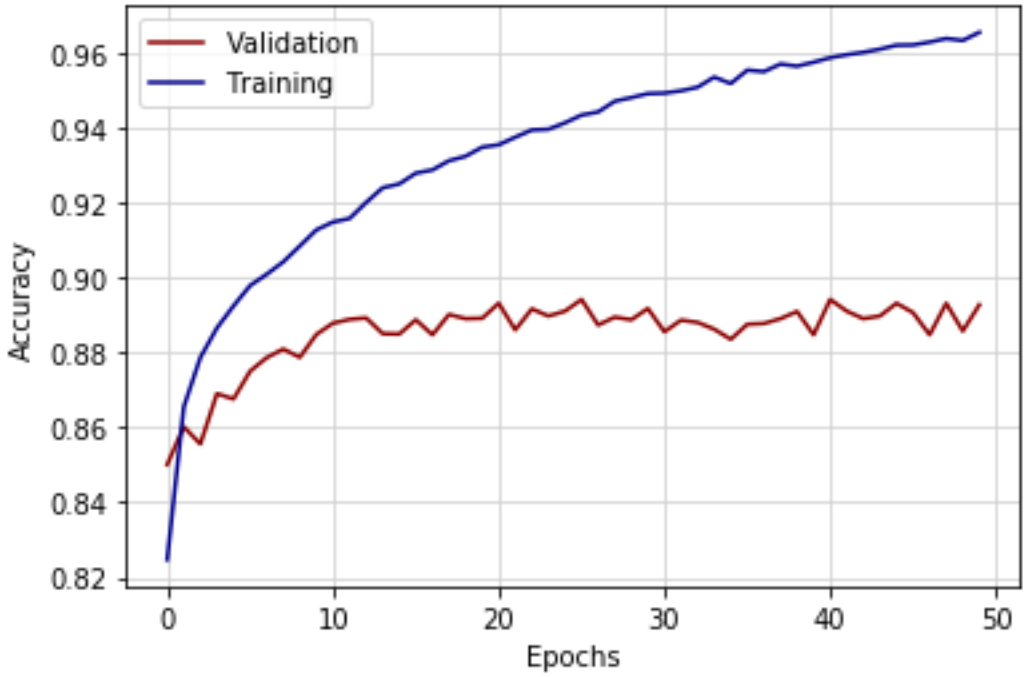
برای سه شبکه ساده، از اپتیمایزر Adam و تابع خطا Cross-entropy استفاده شده است. همچنین تعداد ایپاک ۵۰ و Batch-size=32 در نظر گرفته شده است. درنهایت نمودار دقت داده‌های Validation و Train برحسب تعداد ایپاک برای هر شبکه رسم گردید که در ادامه آمده است.



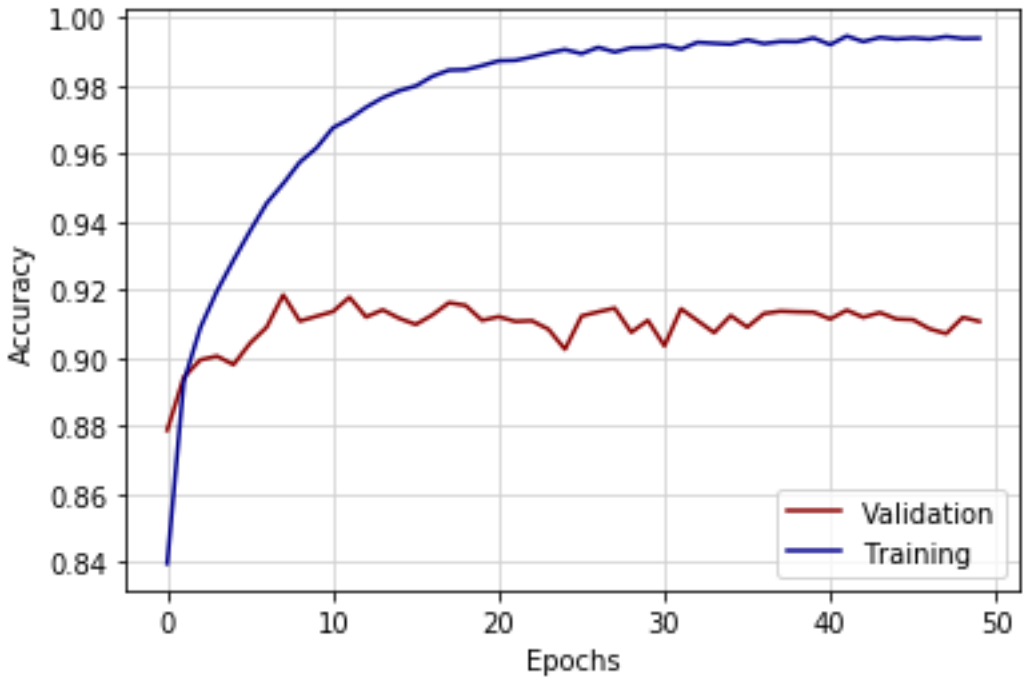
شکل ۹ – نمودار دقت برحسب ایپاک برای شبکه MLP بدون لایه مخفی



شکل ۱۰ – نمودار دقت برحسب ایپاک برای شبکه MLP با یک لایه مخفی



شکل ۱۱ – نمودار دقت برحسب ایپاک برای شبکه MLP با دو لایه مخفی



شکل ۱۲ – نمودار دقت برحسب ایپاک برای شبکه CNN

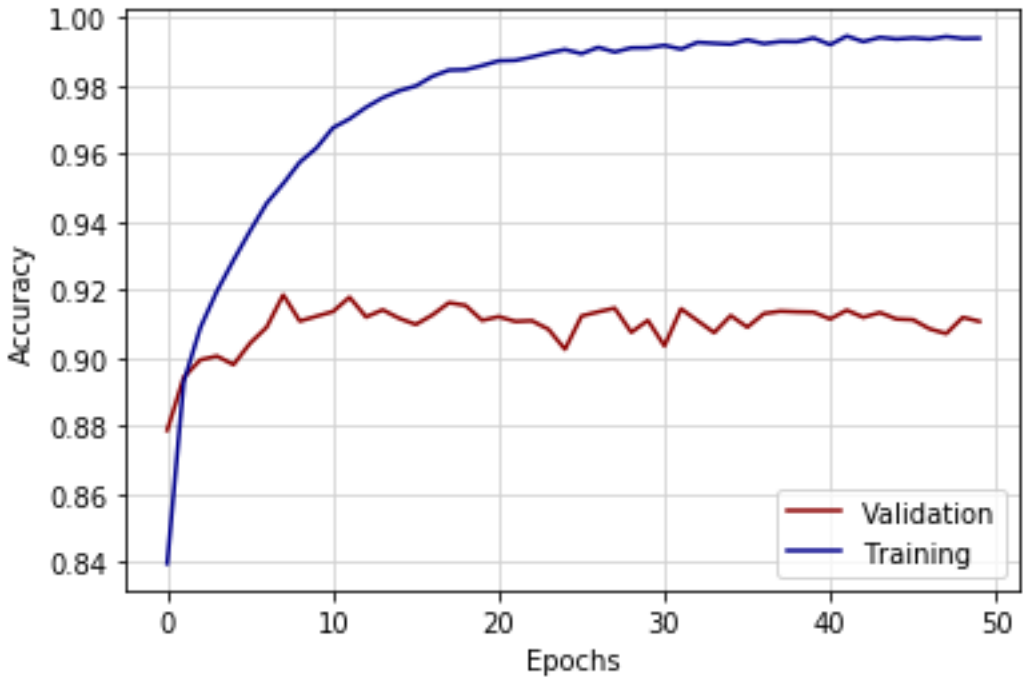
حال براساس نتایج نمودار دقت داده‌های Validation بر تعداد ایپاک برای شبکه‌های مذکور، جدول زیر طراحی گردید. مشاهده می‌شود که دقت داده‌های validation در شبکه CNN بیش‌تر از بقیه است.

جدول ۱ – مقایسه شبکه‌های طراحی شده از منظر دقت داده‌های Validation

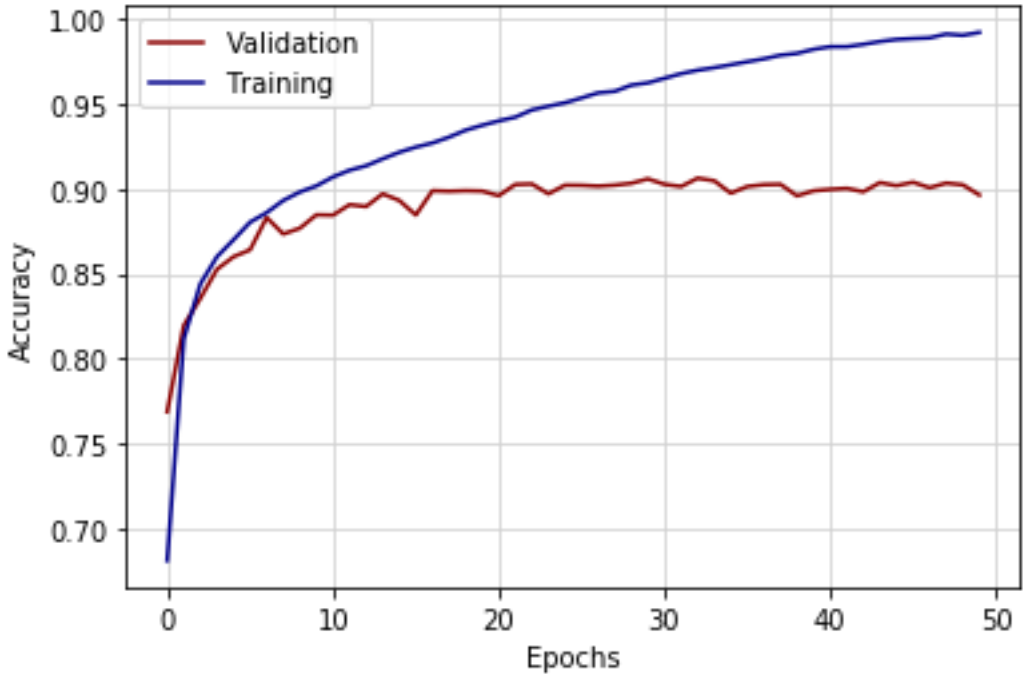
Network	Validation Accuracy	Run-time
MLP Without Hidden Layer	84.43%	**374sec**
MLP With One Hidden Layer	88.78%	376sec
MLP With Two Hidden Layer	89.27%	470sec
CNN	**91.08%**	410sec

۳,۲ تأثیر توابع فعال‌ساز بر شبکه طراحی‌شده

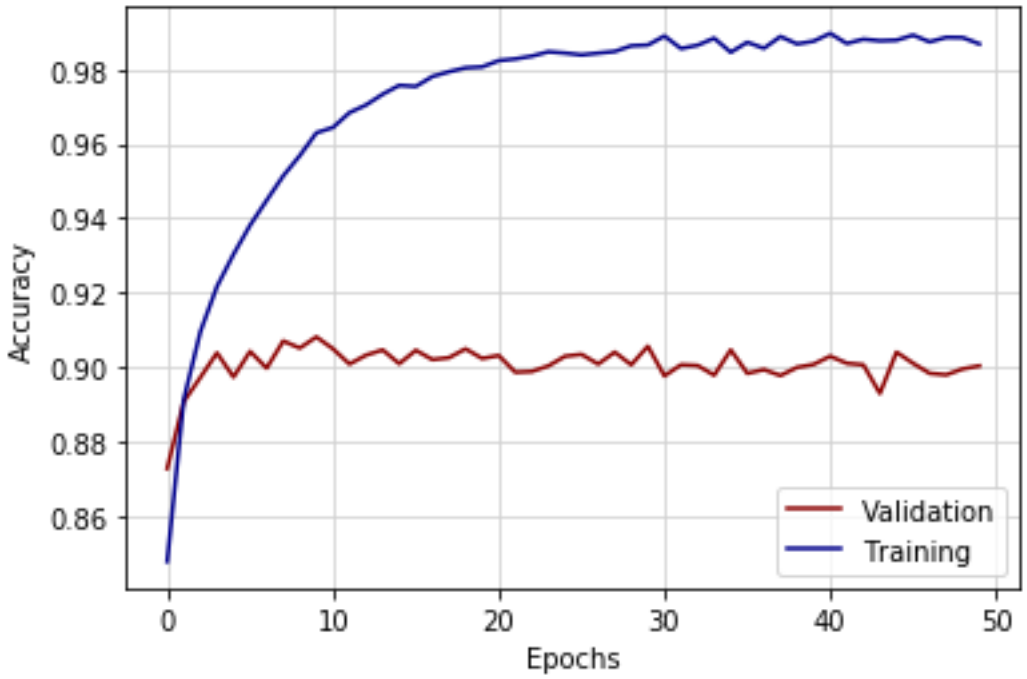
در این بخش، تأثیر توابع فعال‌ساز بررسی شده در بخش اول بر روی شبکه CNN بررسی می‌گردد. البته تابع فعال‌ساز در لایه خروجی در هر سه شبکه، با توابع ReLU، Sigmiod و Tanh، Softmax خواهد بود. پیش‌تر گفته شد که توابع ReLU و Tanh فقط برای لایه‌های میانی مناسب هستند و تابع Sigmoid اگر خروجی دو کلاس داشته باشد به کار می‌رود. پس اعمال توابع فعال‌ساز مختلف بر شبکه CNN نتایج زیر مشاهده گردید.



شکل ۱۳ – نمودار دقت برحسب ایپاک برای شبکه CNN با تابع فعال‌ساز ReLU



شکل ۱۴ – نمودار دقت برحسب ایپاک برای شبکه CNN با تابع فعال‌ساز Sigmoid



شکل ۱۵ – نمودار دقت برحسب ایپاک برای شبکه CNN با تابع فعال‌ساز Tanh

همان‌طور که در نمودارها مشخص است، دقت شبکه برای داده‌های Validation با استفاده از تابع فعال‌ساز ReLU بالاتر از دو حالت دیگر است. در قسمت قبل، مزایا و معایب توابع فعال‌ساز بیان شد. توابع Sigmoid و Tanh با مشکل Gradient Vanishing همراه هستند؛ درواقع با کوچک شدن گرادیان آن‌ها، به‌روزشدن پارامترها متوقف می‌گردد. حال که استفاده از تابع ReLU منجر می‌شود برخی از نورون‌ها در فرایند آموزش غیرفعال و شبکه سبک‌تر شود. براساس دقت داده‌های Validation در ایپاک پنجاهم یک جدول طراحی شده که در ادامه آمده است. همان‌طور که انتظار می‌رفت دقت شبکه با استفاده از تابع فعال‌ساز ReLU بیشتر از سایرین است. بین دو تابع Sigmoid و Tanh، تابع Tanh به دلیل داشتن گرادیان بزرگ‌تر دارای دقت بیش‌تری است.

جدول ۲ – مقایسه تأثیر تابع فعال‌ساز بر دقت داده‌های Validation

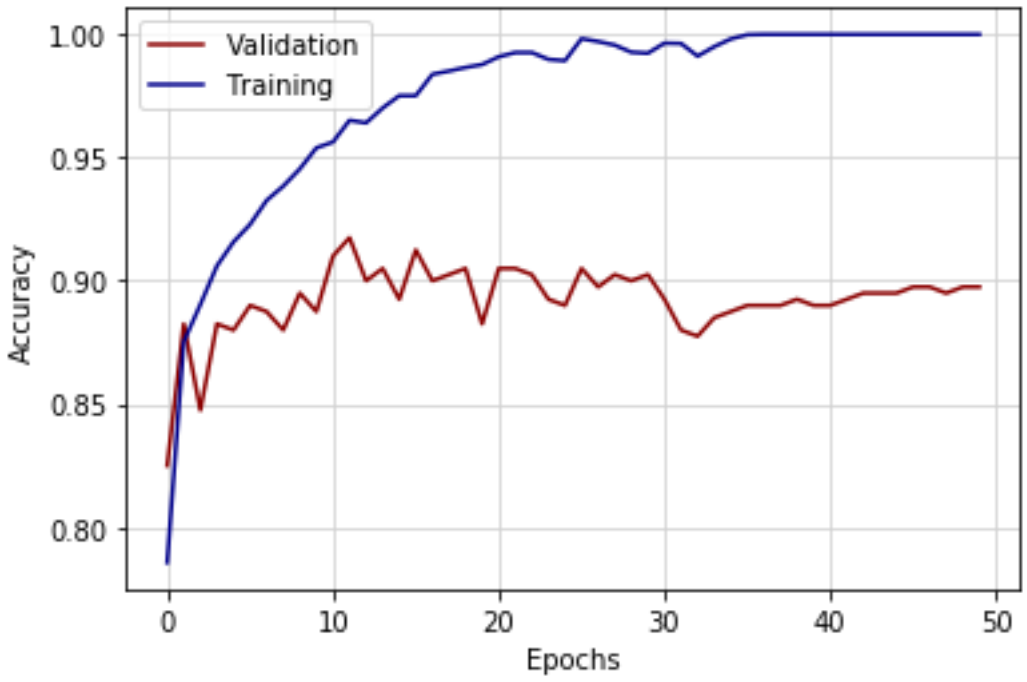
Activation Function	Validation Accuracy
ReLU	**91.08%**
Sigmiod	89.65%
Tanh	90.03%

۳,۳ بررسی تأثیر کاهش نمونه‌ها بر شبکه CNN طراحی‌شده

همان‌طور که گفته شد دیتاست Fashin MNIST شامل ۷۰۰۰۰ داده است که در ده کلاس طبقه‌بندی شده‌اند. در این حین ۶۰۰۰۰ داده مربوط به داده‌های Train و ۱۰۰۰۰ مربوط به داده‌های Test هستند. هدف از این بخش، کاهش نمونه‌های ورودی است به‌گونه‌ای که هر ۸۰۰ نمونه مربوط به یک کلاس باشند. بدیهی است دیتاست جدید شامل ۸۰۰۰ نمونه خواهد بود.

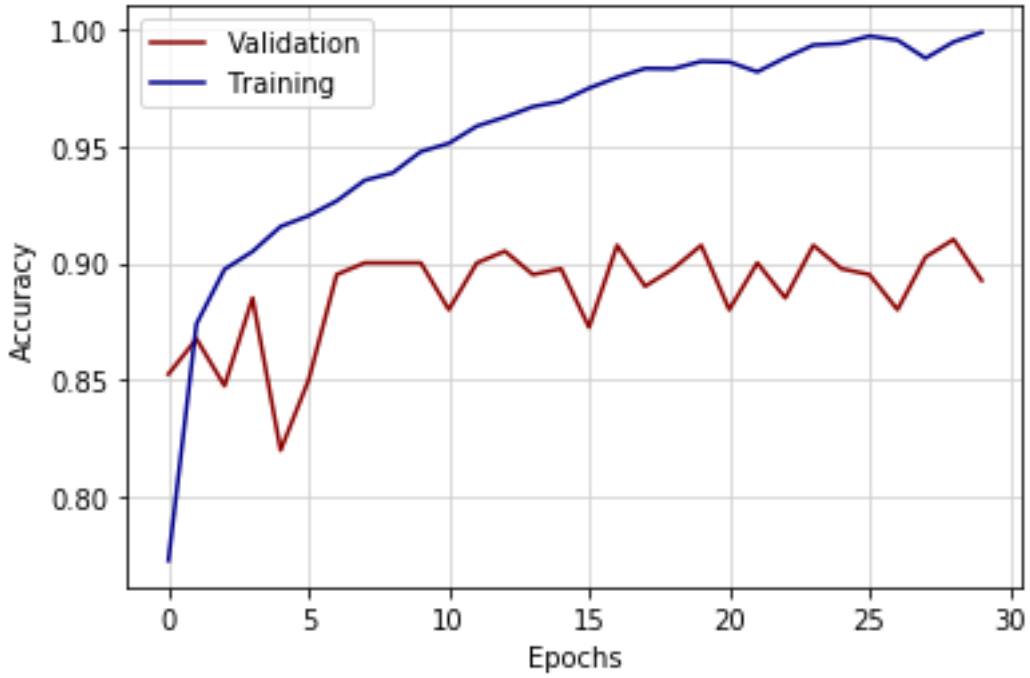
برای کاهش دیتاست شمارنده‌هایی برای هر کلاس در نظر گرفته شد تا تعداد داده‌ها دقیقن ۸۰۰ باشد. بنابراین الگوریتم طراحی شده، هر نمونه را فراخوانی می‌کند. با استفاده از Label هر نمونه، با بررسی یک شرط آن را به کلاس متناظر اضافه می‌کند. شرط گفته‌شده بررسی می‌کند که اگر تعداد اعضای هر کلاس کمتر از ۸۰۰ بود، نمونه را به کلاس مربوطه اضافه می‌کند.

درنهایت دیتاست جدید به شبکه CNN طراحی‌شده اعمال گردید و با همان پارامترهای قبلی شبکه آموزش داده شد. لازم به ذکر است ۵ درصد از دیتاست جدید به‌عنوان داده‌های Validation در نظر گرفته شد. پس از آموزش شبکه در ۵۰ ایپاک، دقت داده‌های Training ۱۰۰ درصد و دقت داده‌های Validation به ۸۹,۲۵ درصد رسید. همچنین نمودار دقت برحسب تعداد ایپاک به‌صورت زیر مشاهده گردید.



شکل ۱۶ – نمودار دقت در ۵۰ ایپاک برای دیتاست کاهش‌یافته

همان‌طور که در شکل مشخص است، دقت داده‌های Train از ایپاک ۳۰ به ۱۰۰ درصد میل می‌کنند و شبکه به‌اصطلاح Overfit شده است. بدین منظور بار دیگر شبکه با ایپاک ۳۰ آموزش داده‌شد. در این حالت دقت داده‌های Train، ۹۹,۸۴ درصد و برای داده‌های Validation ۸۹,۲۵ درصد شد. نمودار دقت برحسب تعداد ایپاک در ادامه آمده است.



شکل ۱۷ – نمودار دقت در ۳۰ ایپاک برای دیتاست کاهش‌یافته

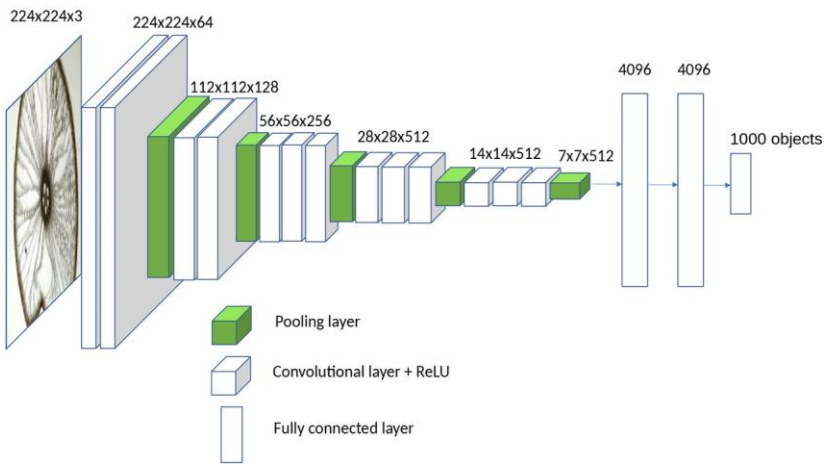
۳ بخش Transfer Learning

۱,۳ معماری VGGNet

در هنگام آموزش شبکه، ورودی لایه Conv اول، تصاویر سه‌کاناله RGB با سایز ۲۲۴*۲۲۴ هستند. تنها پیش‌پردازش انجام شده، تفریق میانگین هر پیکسل برای سه کانال، از مقادیر اصلی آن‌ها است. سپس تصاویر به لایه‌های CONV اعمال می‌شود و با استفاده از فیلترها ویژگی‌های تصاویر استخراج می‌گردند. سایز استفاده شده برای فیلترها ۳*۳ است. البته در یکی از پیکربندی‌های از فیلترهای ۱*۱ نیز استفاده شده است. فیلترهای ۱*۱ به‌عنوان یک انتقال خطی از ورودی به Feature Map استفاده می‌شوند. درواقع هر فیلتر بر روی Patch متناظر از ورودی قرار گرفته و پس از محاسبه مقدار خانه مربوطه در Feature Map، بر روی ورودی Sweep می‌شود. در معماری VGG سوییپ به اندازه یک پیکسل است (stride=1). همچنین ابعاد تصاویر پس از اعمال فیلترها، حفظ می‌شود؛ بنابراین از Padding به اندازه ۱ پیکسل استفاده شده است. از ۵ لایه Max-pooling با ابعاد ۲*۲ و با Stride=2 نیز استفاده شده است. پس از لایه‌های Conv و Pooling از سه لایه Fully-connected استفاده شده است. دو لایه Fully-connected اول ۴۰۹۶ نورون دارند. لایه آخر براساس استاندارد مسابقه ILSVRC که برای دسته‌بندی ۱۰۰۰ کلاس بوده، ۱۰۰۰ نورون داشته و تابع فعال‌ساز Softmax استفاده کرده است. سایر لایه‌ها، CONV، Max-pooling و Dense، از تابع فعال‌ساز ReLU بهره برده‌اند. تعداد کانال‌های استفاده شده در لایه‌های CONV کوچک است؛ به‌گونه‌ای که در اولین لایه از ۶۴ فیلتر استفاده شده و بعد از هر Max-pooling دوبرابر گردیده تا به ۵۱۲ فیلتر رسیده است. پیکربندی‌های VGG عمق متفاوتی دارند؛ عمق پیکربندی‌ها بین ۱۱ تا ۱۹ لایه وزن‌دار است. به‌عنوان‌مثال در VGG16، ۱۳۴ میلیون پارامتر آموزش داده می‌شود.

در معماری VGG از الگوریتم بهینه‌سازی Mini-batch Gradient Descent با Momentum استفاده شده است؛ به‌گونه‌ای‌که مقادیر Batch-size و Momentum، به ترتیب ۲۵۶ و ۰,۹ انتخاب شده‌اند. برای رگولاریزیشن از Wight Decay (L2) استفاده شده است. همچنین دو لایه ابتدایی Fully-connected از Dropout با میزان ۵۰ درصد بهره برده‌اند.

درابتدا از نرخ یادگیری ۰,۰۱ استفاده شده است؛ اما بدلیل موقف شدن افزایش دقت داده‌های Validation، نرخ یادگیری ۰,۰۰۱ شده است. به‌طورکلی نرخ یادگیری سه بار کاهش یافته است. درنهایت فرایند یادگیری پس از ۳۷۰۰۰۰ تکرار (۷۴ ایپاک) متوقف گردید.



شکل – معماری VGGNet

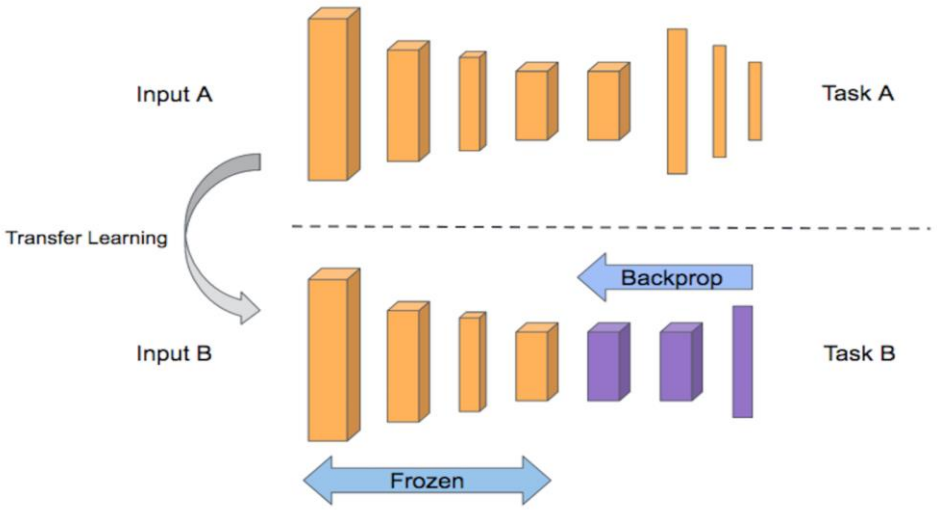
از مزایای VGG می‌توان به ساده و یک‌پارچگی آن اشاره کرد. درواقع در این نوع معماری از فیلترهای کوچک ولی با تعداد تکرار بیشتر استفاده شده است. معماری VGG از این نکته استفاده کرده است که دو فیلتر کانولوشنی ۳*۳ معادل یک فیلتر ۵*۵ و سه فیلتر ۳*۳ معادل یک فیلتر ۷*۷ است. بنابراین مزیت اصلی VGG، استفاده از دو ۳*۳ به جای یک ۵*۵ و سه فیلتر ۳*۳ به جای یک ۷*۷ است که افزایش غیرخطینگی را به دنبال دارد. زیرا هنگام استفاده از هر فیلتر یک تابع فعال‌ساز ReLU استفاده می‌شود. حال با افزایش تعداد فیلترها، از توابع ReLU بیش‌تری نیز استفاده می‌شود که منجر به افزایش Non-linearity شده و درواقع افزایش Non-linearity افزایش قدرت تعمیم شبکه و دقت داده‌های validation را به دنبال دارد.

معماری VGGNet با افزودن عمق به شبکه، توانست در سال ۲۰۱۴ با استفاده از بخشی از دیتا ImageNet (نه همه آن)، توانست به دقت بالاتری نسبت به رقبا خود دست پیدا کند. عمده کاربرد VGGNet در مسائل Classification است؛ خصوصاً هنگامی‌که دیتاست برای مسئله مدنظر کوچک است از شبکه VGGNet به‌عنوان Pre-train استفاده می‌شود. در بخش‌های آتی درمورد Transfer Learning و Pre-training صحبت می‌شود.

۲,۳ Transfer Learning

فرایند یادگیری شبکه‌های عصبی عمیق برای یک دیتاست بزرگ زمان‌بر بوده و ممکن است روزها یا هفته‌ها طول بکشد. یک راه میانبر استفاده مجدد از مدل‌های از پیش آموزش داده شده همانند مدل‌های تشخیص تصاویر ImageNet، برای دیتاست مدنظر است. درواقع Transfer Learning، فرایندی است که شبکه عصبی از یک

Task یاد گرفته و بر روی Task مشابه دیگر پیاده می‌شود؛ به‌گونه‌ای که با افزودن یک یا دو لایه، شبکه اولیه برای مسئله دوم قابل‌استفاده می‌گردد. از فواید Transfer Learning می‌توان به کاهش زمان آموزش اشاره کرد. همچنین ازآنجایی‌که شبکه برای یک دیتاست اولیه آموزش داده شده، از قدرت تعمیم بالایی برخوردار است.



شکل ۱۸ – مفهوم Transfer Learning

یکی از دلایل استفاده از Transfer Learning آن است که دیتاست موجود کوچک است. بنابراین از سیستمی که با داده‌های زیاد Train شده استفاده و دیتاست کوچک را بر آن اعمال می‌شود. به‌عنوان‌مثال یک شبکه برای یک میلیون عکس گربه آموزش داده شده و برای دیتاست صدتایی رادیولوژی استفاده می‌شود. دلیل کارآمد بودن Transfer Learning آن است که بیشتر Low-level Feature ها همانند تشخیص لبه، منحنی، اشیا مثبت و ... در دیتاست اول، منجر به عملکرد بهتر در دیتاست دوم می‌شود. درواقع می‌توان از ویژگی‌های شبکه Pre-train، در شبکه مدنظر نیز بهره برد. البته نمی‌توان از شبکه‌ای استفاده کرد که داده‌های Pre-train کمتری نسبت به دیتاست مدنظر دارد.

ازطرفی آموزش شبکه‌هایی همچون VGGNet، ResNet و ... با امکان دسترسی به GPU قدرتمند، در حدود چند هفته طول کشیده است. پس دلیل بعدی برای استفاده از Transfer Lerning می‌تواند نبود امکانات و کم بودن زمان و هزینه باشد.

۳,۳ پیاده‌سازی VGG16 با استفاده از Transfer Learning

در این بخش با استفاده از فریم‌ورک کراس، معماری VGG16 فراخوانی گردید؛ به‌گونه‌ای‌که برای قسمت‌های بعد از وزن‌های شبکه VGG استفاده می‌شود. درقسمت‌های بعدی با جزییات بیش‌تر این قسمت توضیح داده می‌شود.

۴,۳ لیست اشیا قابل‌شناسایی برای VGG16

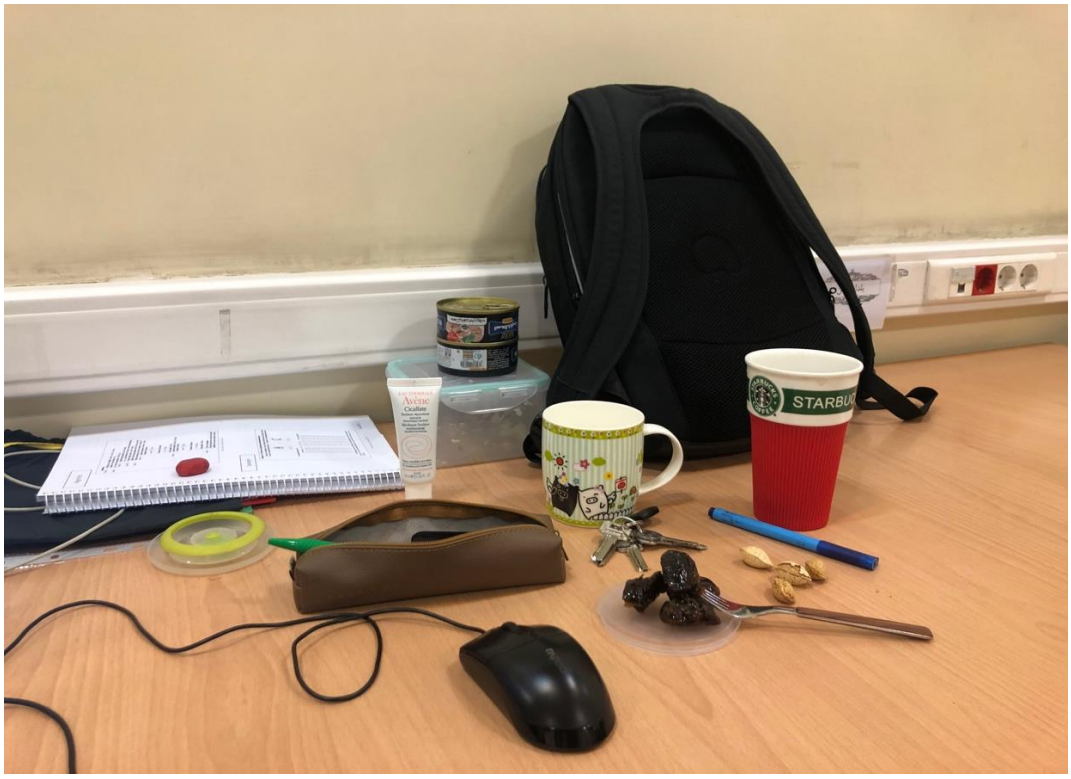
برای دیدن لیست اشیایی که معماری VGG می‌تواند شناسایی کند، یا همان هزار دسته مربوط به ImageNet، از یک حقه استفاده شده است! به‌گونه‌ای که پس از فراخوانی VGG16 موجود در Keras، یک عکس گربه وارد گردید. به جای اختصاص دادن نزدیک‌ترین کلاس به تصویر، ۱۰۰۰ کلاس نزدیک مشاهده گردید. بدین‌صورت همه دسته‌های موجود در VGGNet مشاهده گردید. در ادامه اسامی این کلاس‌ها آمده است. همچنین یک فایلtxt. همراه کد این بخش آمده که اسامی اشیا قابل تشخیص معماری VGG در آن ذخیره گردیده‌است.

```
Egyptian_cat, tabby, tiger_cat, lynx, bow_tie, carton, radiator, vase, remote_control, ping-pong_ball, space_heater, window_screen, bath_towel, punching_bag, velvet, cup, mouse, pitcher, tub,
feather_boa, lens_cap, sock, paper_towel, doormat, refrigerator, washer, washbasin, window_shade, lampshade, wall_clock, table_lamp, candle, crossword_puzzle, bucket, bathtub, goblet, quilt, bonnet,
coffee_mug, plastic_bag, bookcase, cowboy_hat, iPod, water_bottle, toilet_tissue, fountain_pen, mailbag, Christmas_stocking, screen, shower_curtain, computer_keyboard, red_wine, pillow, swab, spotlight,
jean, drumstick, fur_coat, wooden_spoon, Siamese_cat, plunger, cowboy_boot, rubber_eraser, hook, scale, hammer, paintbrush, television, pool_table, syringe, notebook, hard_disc, mixing_bowl, monitor,
beer_bottle, iron, printer, dishwasher, saltshaker, great_grey_owl, soap_dispenser, pajama, frying_pan, shower_cap, maillot, cellular_telephone, spatula, mitten, broom, purse, cloak, brassiere, cleaver,
power_drill, hatchet, basenji, beer_glass, studio_couch, buckle, lighter, dumbbell, thimble, barometer, sombrero, hamper, letter_opener, screwdriver, switch, caldron, laptop, Crock_Pot, rule,
toilet_seat, miniskirt, perfume, wine_bottle, hair_spray, stole, water_jug, photocopier, microwave, wok, desktop_computer, bottlecap, wig, medicine_chest, radio, backpack, teapot, hummingbird,
pineapple, sweatshirt, umbrella, ice_lolly, tiger, hourglass, apron, ear, matchstick, pop_bottle, gown, suit, Persian_cat, tennis_ball, can_opener, ski_mask, pill_bottle, necklace, mousetrap, knot,
Windsor_tie, broccoli, mask, screw, abacus, sliding_door, ashcan, lipstick, shoji, crate, bookshop, panpipe, mortarboard, ballpoint, vacuum, hand_blower, leopard, wallet, corn, banana, sarong, conch,
balance_beam, maillot, prayer_rug, harmonica, sleeping_bag, packet, ladle, hare, mortar, pinwheel, armadillo, bow, cougar, eggnog, fox_squirrel, Boston_bull, soup_bowl, military_uniform, wardrobe,
book_jacket, wood_rabbit, acoustic_guitar, Yorkshire_terrier, theater_curtain, Norwich_terrier, pedestal, Ibizan_hound, bib, bannister, jack-o'-lantern, spindle, coil, standard_schnauzer, Band_Aid,
miniature_pinscher, projector, tray, loupe, jigsaw_puzzle, safety_pin, zucchini, horned_viper, meerkat, stethoscope, wool, cardigan, Doberman, analog_clock, modem, corkscrew, prairie_chicken,
comic_book, kimono, slide_rule, slug, alligator_lizard, totem_pole, mantis, Italian_greyhound, altar, cock, sewing_machine, lotion, marmoset, poncho, torch, sunglasses, toaster, centipede, loudspeaker,
trench_coat, running_shoe, triceratops, dishrag, crib, Weimaraner, zebra, Chihuahua, piggy_bank, file, beaker, envelope, four-poster, violin, sunscreen, breastplate, coffeepot, kelpie, bubble,
seat_belt, chime, miniature_schnauzer, maraca, grand_piano, motor_scooter, Eskimo_dog, pick, binder, microphone, orange, chain, pole, revolver, parallel_bars, bikini, cicada, barrel, fountain, nail,
giant_schnauzer, ruffed_grouse, crayfish, shopping_basket, bolo_tie, silky_terrier, teddy, strainer, cocktail_shaker, balloon, gong, throne, vestment, reflex_camera, nipple, grasshopper, vizsla,
mailbox, diaper, cassette, robin, stove, library, Angora, sundial, hay, French_bulldog, home_theater, butternut_squash, rain_barrel, pencil_box, dalmatian, scabbard, bearskin, barbell, tile_roof,
digital_clock, quill, horizontal_bar, handkerchief, jaguar, hip, face_powder, car_mirror, German_short-haired_pointer, dial_telephone, joystick, rocking_chair, academic_gown, abaya, wombat, tripod,
walking_stick, toy_terrier, kite, electric_guitar, baseball, Scotch_terrier, knee_pad, bull_mastiff, projectile, dining_table, Airedale, shield, pot, mongoose, gar, hamster, crutch, overskirt,
birdhouse, green_lizard, snow_leopard, black-footed_ferret, Polaroid_camera, plate_rack, drum, lacewing, swimming_trunks, flute, burrito, cuirass, sunglass, shovel, espresso, acorn, frilled_lizard,
bluetick, digital_watch, chain_mail, fire_screen, honeycomb, measuring_cup, traffic_light, desk, Siberian_husky, magnetic_compass, barrow, flagpole, Granny_Smith, ice_cream, Australian_terrier,
French_horn, gasmask, banjo, barbershop, hoopskirt, soccer_ball, partridge, upright, grey_fox, junco, snail, electric_fan, wallaby, box_turtle, Norwegian_elkhound, long-horned_beetle, cricket,
oscilloscope, coyote, terrapin, lycaenid, cradle, lab_coat, stopwatch, tobacco_shop, chocolate_sauce, bassinet, American_Staffordshire_terrier, titi, weasel, pickelhaube, cocker_spaniel, dough,
Great_Dane, cucumber, bittern, house_finch, ringlet, bassoon, chest, cinema, sandal, picket_fence, oboe, butcher_shop, basset, folding_chair, Brittany_spaniel, pretzel, waffle_iron, unicycle, whiptail,
schipperke, church, swing, padlock, African_chameleon, admiral, sea_lion, redbone, combination_lock, Norfolk_terrier, pencil_sharpener, agama, neck_brace, safe, web_site, boxer, mink, African_grey,
plate, hognose_snake, lemon, muzzle, espresso_maker, chambered_nautilus, guacamole, restaurant, Bouvier_des_Flandres, clog, holster, barracouta, rugby_ball, mosque, Rottweiler, Mexican_hairless,
crash_helmet, common_iguana, spider_web, maze, German_shepherd, obelisk, jersey, puffer, Cardigan, vine_snake, beagle, American_chameleon, hand-held_computer, fly, cornet, tricycle, milk_can, coho,
hermit_crab, shoe_shop, Dungeness_crab, spiny_lobster, hen, barn_spider, tape_player, pomegranate, bathing_cap, night_snake, CD_player, wolf_spider, ocarina, grille, Kerry_blue_terrier, peacock, cello,
reel, American_lobster, parking_meter, marimba, sax, crane, platypus, head_cabbage, moped, bell_pepper, Lakeland_terrier, carpenter's_kit, water_ouzel, whistle, racket, stretcher, black-and-
tan_coonhound, fig, toy_poodle, street_sign, car_wheel, starfish, strawberry, chimpanzee, buckeye, ballplayer, paddle, entertainment_center, bloodhound, brass, pizza, Labrador_retriever, harp, carousel,
nematode, hair_slide, Loafer, prison, West_Highland_white_terrier, Border_terrier, macaque, polecat, guillotine, cassette_player, chiffonier, diamondback, rifle, electric_ray, missile, potter's_wheel,
stupa, black_grouse, grocery_store, black_and_gold_garden_spider, cockroach, guenon, cab, jay, sulphur_butterfly, Rhodesian_ridgeback, trombone, lion, guinea_pig, vault, lorikeet, accordion,
cauliflower, ibex, cheetah, sidewinder, jacamar, oil_filter, pug, bustard, basketball, Welsh_springer_spaniel, macaw, trilobite, cannon, otter, oxygen_mask, confectionery, bee, malinois, French_loaf,
bulletproof_vest, albatross, china_cabinet, redshank, Gordon_setter, tick, Border_collie, Appenzeller, typewriter_keyboard, whippet, hotdog, banded_gecko, timber_wolf, Chesapeake_Bay_retriever,
greenhouse, dome, quail, malamute, tree_frog, cabbage_butterfly, potpie, streetcar, chainlink_fence, Bernese_mountain_dog, menu, turnstile, barber_chair, Madagascar_cat, bulbul, triumphal_arch,
space_bar, goldfish, stage, king_penguin, whiskey_jug, vulture, mud_turtle, mashed_potato, vending_machine, magpie, skunk, jackfruit, Pembroke, indri, Old_English_sheepdog, carbonara, squirrel_monkey,
red_fox, sea_urchin, boa_constrictor, dingo, bakery, cash_machine, kit_fox, spaghetti_squash, king_crab, lawn_mower, tiger_shark, shopping_cart, castle, English_springer, pirate, bagel, consomme,
dragonfly, gazelle, palace, green_mamba, affenpinscher, hartebeest, binoculars, patio, red_wolf, rotisserie, scorpion, harvestman, koala, garden_spider, red-breasted_merganser, half_track, leafhopper,
red-backed_sandpiper, anemone_fish, brambling, Staffordshire_bullterrier, wire-haired_fox_terrier, football_helmet, orangutan, sea_slug, Irish_water_spaniel, thatch, schooner, rock_beauty, trifle,
collie, acorn_squash, ladybug, Bedlington_terrier, siamang, space_shuttle, cardoon, container_ship, black_widow, yellow_lady's_slipper, flat-coated_retriever, badger, puck, toucan, English_setter,
Pekinese, king_snake, worm_fence, drake, mosquito_net, goose, Greater_Swiss_Mountain_dog, flatworm, Shih-Tzu, Indian_cobra, hot_pot, stone_wall, mushroom, thunder_snake, park_bench, golf_ball, cairn,
liner, dock, golden_retriever, meat_loaf, Petri_dish, ski, soft-coated_wheaten_terrier, great_white_shark, Blenheim_spaniel, organ, pay-phone, isopod, hyena, gas_pump, sports_car, monastery, yawl, pier,
rhinoceros_beetle, toyshop, Brabancon_griffon, leatherback_turtle, Irish_terrier, hornbill, bullet_train, monarch, bald_eagle, sea_cucumber, Sussex_spaniel, assault_rifle, wing, African_crocodile,
custard_apple, daisy, proboscis_monkey, cheeseburger, bell_cote, recreational_vehicle, tank, miniature_poodle, groenendael, otterhound, hammerhead, limousine, papillon, Lhasa, bee_eater, Walker_hound,
coral_fungus, rock_python, English_foxhound, parachute, artichoke, amphibian, trailer_truck, forklift, jinrikisha, viaduct, lionfish, water_snake, manhole_cover, impala, croquet_ball, black_swan, eel,
garter_snake, Gila_monster, ringneck_snake, bullfrog, Scottish_deerhound, Dutch_oven, gorilla, curly-coated_retriever, Japanese_spaniel, jeep, Pomeranian, beaver, weevil, sturgeon, Saint_Bernard,
Irish_setter, European_gallinule, seashore, earthstar, snorkel, freight_car, green_snake, groom, suspension_bridge, water_tower, pickup, Shetland_sheepdog, gyromitra, stinkhorn, tarantula, megalith,
Great_Pyrenees, lesser_panda, ambulance, odometer, tailed_frog, lumbermill, ant, solar_dish, tractor, capuchin, minibus, canoe, American_alligator, Irish_wolfhound, common_newt, minivan, moving_van,
brain_coral, Sealyham_terrier, gondola, American_black_bear, goldfinch, hippopotamus, school_bus, chain_saw, steel_drum, damselfly, coucal, snowplow, EntleBucher, tench, maypole, porcupine, axolotl,
beach_wagon, American_egret, American_coot, Model_T, brown_bear, slot, volleyball, borzoi, mobile_home, Newfoundland, three-toed_sloth, Samoyed, clumber, warplane, oystercatcher, chow, garbage_truck,
trimaran, dowitcher, dhole, ox, racer, Maltese_dog, ground_beetle, ptarmigan, convertible, Tibetan_terrier, black_stork, wild_boar, beacon, sea_anemone, patas, geyser, bobsled, plane, langur, agaric,
chiton, European_fire_salamander, rock_crab, bighorn, barn, ostrich, fire_engine, ruddy_turnstone, disk_brake, howler_monkey, crane, golfcart, cliff, eft, tow_truck, airship, catamaran,
African_hunting_dog, standard_poodle, rapeseed, hog, sorrel, baboon, horse_cart, Saluki, pelican, tiger_beetle, radio_telescope, limpkin, Arctic_fox, wreck, volcano, chickadee, fireboat, marmot, kuvasz,
hen-of-the-woods, flamingo, spoonbill, passenger_car, breakwater, police_van, komondor, aircraft_carrier, scoreboard, lifeboat, gibbon, white_wolf, planetarium, bolete, echidna, African_elephant, llama,
Afghan_hound, keeshond, Indian_elephant, ram, loggerhead, coral_reef, lakeside, jellyfish, sloth_bear, grey_whale, killer_whale, indigo_bunting, airliner, Komodo_dragon, promontory, spotted_salamander,
briard, stingray, alp, Leonberg, scuba_diver, harvester, little_blue_heron, sea_snake, submarine, oxcart, dogsled, leaf_beetle, Dandie_Dinmont, spider_monkey, trolleybus, speedboat, paddlewheel,
bicycle-built-for-two, dugong, drilling_platform, apiary, boathouse, cliff_dwelling, mountain_bike, Tibetan_mastiff, sandbar, electric_locomotive, sulphur-crested_cockatoo, valley, ice_bear, dam,
colobus, steam_locomotive, plow, giant_panda, snowmobile, mountain_tent, bison, white_stork, warthog, steel_arch_bridge, water_buffalo, fiddler_crab, thresher, go-kart, Arabian_camel, dung_beetle,
tusker, yurt,
```

شکل ۱۹ – لیست اشیا قابل‌شناسایی برای معماری VGG

۵,۳ فراخوانی یک تصویر و تشخیص اشیا

ابعاد تصویر گرفته شده ۳۰۲۴*۴۰۳۲ پیکسل در پیکسل است. در ابتدا ابعاد تصویر به ۲۴۴*۲۴۴ تبدیل می‌شود تا بتوان به‌عنوان ورودی به شبکه VGG16 اعمال کرد.



شکل ۲۰ – تصویر اعمال‌شده به شبکه VGG16

تصویر به فرمت PIL است و برای اعمال آن به شبکه عمیق مدنظر باید فرمت آن به Numpy Array تبدیل گردد. بدین منظور از تابع img_to_array() استفاده می‌شود. حال لازم است به شبکه تعداد نمونه‌ها اعمال گردد؛ ازآنجایی‌که از یک تصویر استفاده شده است، فرم ورودی به شکل (1,224,224,3) درمی‌آید. سپس با استفاده از شبکه اپلیکیشن‌ها موجود در Keras، مدل VGG16 فراخوانی شده و خلاصه‌ای از آن مشاهده می‌گردد. همان‌طور که انتظار می‌رفت این شبکه در حدود ۱۴۰ میلیون پارامتر را به‌روزرسانی کرده‌است.

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_9 (InputLayer)         (None, 224, 224, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              102764544
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
_________________________________________________________________
predictions (Dense)          (None, 1000)              4097000
=================================================================
Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
```

شکل ۲۱ – خلاصه‌ای از مدل VGG16

درنهایت پیکسل‌ها با تابع preprocess_input پیش‌پردازش می‌شوند و به مدل VGG16 اعمال می‌گردند. همچنین با تابع decode_prediction سه نمونه از بیشترین احتمالات موجود در عکس، مشاهده می‌گردد که برای عکس مذکور، میز، ماوس و کوله‌پشتی تشخیص داده شده است. لازم به ذکر است اعداد جلو اسامی، درصد احتمال آن‌ها است.

```
[('desk', 73.25401902198792), ('mouse', 4.669767618179321), ('backpack', 2.305394969880581)]
```

شکل ۲۲ – تشخیص سه اشیا با بیش‌ترین احتمال

منابع و مآخذ

1. https://www.coursera.org/specializations/deep-learning
2. https://www.coursera.org/specializations/tensorflow-in-practice?
3. Optimizing Gradient Descent- http://sebastianruder.com/optimizing-gradient-descent/
4. Dean, J., Corrado, G. S., Monga, R., Chen, K., Devin, M., Le, Q. V, … Ng, A. Y. (2012). Large Scale Distributed Deep Networks. NIPS 2012: Neural Information Processing Systems. http://doi.org/10.1109/ICDAR.2011.95
5. Qian, N. (1999). On the momentum term in gradient descent learning algorithms. Neural Networks: The Official Journal of the International Neural Network Society, 12(1), 145–151. http://doi.org/10.1016/S0893-6080(98)00116-6
6. Kingma, D. P., & Ba, J. L. (2015). Adam: a Method for Stochastic Optimization. International Conference on Learning Representations
7. Zaremba, W., & Sutskever, I. (2014). Learning to Execute, 1–25. Retrieved from http://arxiv.org/abs/1410.4615
8. Zhang, S., Choromanska, A., & LeCun, Y. (2015). Deep learning with Elastic Averaging SGD. Neural Information Processing Systems Conference (NIPS 2015).Retrieved from http://arxiv.org/abs/1412.6651
9. Darken, C., Chang, J., & Moody, J. (1992). Learning rate schedules for faster stochastic gradient search. Neural Networks for Signal Processing II Proceedings of the 1992 IEEE Workshop, (September). http://doi.org/10.1109/NNSP.1992.253713