به نام خدا

دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده مهندسی برق و کامپیوتر

سیستم‌های هوشمند

گزارش تکلیف شماره ۴

حسین عطرسایی ۸۱۰۱۹۸۴۴۲

دی ماه ۱۴۰۱

فهرست

چکیده

هدف از این تکلیف آشنایی با شبکهٔ عصبی و نحوهٔ پیاده سازی آن می‌باشد.

در بخش اول این تکلیف با مفاهیم و روابط تئوری در شبکهٔ عصبی چند لایه آشنا شده و آن را با در پایتون با استفاده از کتابخانهٔ numpy پیاده سازی می‌کنیم.

در بخش دوم به بررسی شبکه‌های عصبی چند لایه و کانوولاشنی پرداخته و با استفاده از مجموعهٔ دادگان CIFAR عملکرد مدل‌های طراحی شده را ارزیابی می‌نماییم.

در نهایت نیز با یادگیری انتقال‌یافته و Efficient Net آشنا شده و آن را در پایتون پیاده‌سازی می‌نماییم.

سوال اول) شبکه عصبی پرسپترون با چند لایهٔ مخفی

الف) تحلیلی

ابتدا پارامترهای داده شده را براساس شماره دانشجویی (۸۱۰۱۹۸۴۴۲) به دست می‌آوریم:

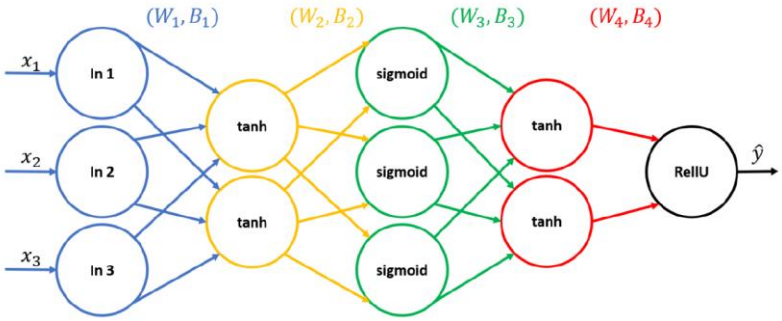
$$X_۱ = \begin{bmatrix} ۲ \\ ۴ \\ ۲ \end{bmatrix}, \quad X_۲ = \begin{bmatrix} ۴ \\ ۲ \\ ۴ \end{bmatrix}, \quad y_۱ = [۲], \quad y_۲ = [۴]$$

$$\widetilde{W}_۱ = W_۱^T = \begin{bmatrix} ۰/۱۲ & ۰/۳۲ & ۰/۵۲ \\ ۰/۲۴ & ۰/۴۴ & ۰/۴۴ \end{bmatrix}, \quad \widetilde{W}_۲ = W_۲^T = \begin{bmatrix} ۲/۱۵ & ۴/۴۵ \\ ۲/۲۵ & ۴/۵۵ \\ ۲/۳۵ & ۴/۶۵ \end{bmatrix}$$

$$\widetilde{W}_۳ = W_۳^T = \begin{bmatrix} ۸/۱۲ & ۸/۳۲ & ۸/۵۲ \\ ۸/۲۲ & ۸/۴۲ & ۸/۶۲ \end{bmatrix}, \quad \widetilde{W}_۴ = W_۴^T = [-۱/۸۴ \quad -۱/۶۴]$$

$$B_۱ = \begin{bmatrix} ۰/۲۱ \\ ۰/۴۲ \end{bmatrix}, \quad B_۲ = \begin{bmatrix} ۶/۱۵ \\ ۶/۲۵ \\ ۶/۳۵ \end{bmatrix}, \quad B_۳ = \begin{bmatrix} ۰/۵۲ \\ ۰/۶۲ \end{bmatrix}, \quad B_۴ = [۲/۲۶]$$

حال مسیر Feed Forward را در شبکهٔ زیر طی کرده و پارامترهای در طول مسیر به دست می‌آوریم.



شکل ۱ـ۱: شبکهٔ عصبی چند لایه

برای راحتی در نام گذاری، خروجی activation functionها را به صورت a_i نام گذاری می‌کنیم. همچنین weighted sumهای هر بخش را نیز با S_i نشان می‌دهیم.

$$a_۱ = Z = \tanh(\widetilde{W}_۱ X_۱ + B_۱) = \tanh(S_۱) = \begin{bmatrix} ۰/۹۲۲ \\ ۰/۹۹۸۳ \end{bmatrix}$$

$$a_۲ = K = \text{sigmoid}(\widetilde{W}_۲ a_۱ + B_۲) = \text{sigmoid}(S_۲) \cong \begin{bmatrix} ۱ \\ ۱ \\ ۱ \end{bmatrix}$$

$$a_۳ = P = \tanh(\widetilde{W}_۳ a_۲ + B_۳) = \tanh(S_۳) \cong \begin{bmatrix} ۱ \\ ۱ \end{bmatrix}$$

$$a_۴ = \hat{y} = \text{ReLU}(\widetilde{W}_۴ a_۳ + B_۴) = \text{ReLU}(S_۴) \cong [۰]$$

$$E = \frac{۱}{۲}(\hat{y} - y_۱)^۲ = ۲$$

حال برای طی کردن مسیر Back Propagation، از قوانین زیر برای آپدیت وزن‌ها استفاده می‌کنیم:

$$\frac{\partial E}{\partial \widetilde{W}_i} = \frac{\partial E}{\partial S_i}(a_{i-1})^T \ , \quad \frac{\partial E}{\partial S_i} = \frac{\partial E}{\partial a_i} \odot \frac{\partial a_i}{S_i} \ , \quad \frac{\partial E}{\partial B_i} = \frac{\partial E}{\partial S_i} \ , \quad \frac{\partial E}{\partial a_{i-1}} = \left(\widetilde{W}_i\right)^T \frac{\partial E}{\partial S_i}$$

$$\frac{\partial E}{\partial \widetilde{W}_4} = \frac{\partial E}{\partial S_4}(a_3)^T = \frac{\partial E}{\partial \hat{y}} \odot \frac{\partial \hat{y}}{\partial S_4}(a_3)^T == (\hat{y} - y)H(S_4)(a_3)^T = [0 \quad 0]$$

$$\frac{\partial E}{\partial B_4} = \frac{\partial E}{\partial S_4} = \frac{\partial E}{\partial a_4} \odot \frac{\partial a_4}{S_4} = (\hat{y} - y) \odot H(S_4) = [0]$$

$$\frac{\partial E}{\partial \widetilde{W}_3} = \frac{\partial E}{\partial S_3}(a_2)^T = \frac{\partial E}{\partial a_3} \odot \frac{\partial a_3}{S_3}(a_2)^T = \left(\widetilde{W}_4\right)^T \frac{\partial E}{\partial S_4} \odot \frac{\partial a_3}{S_3}(a_2)^T$$

$$= [(\hat{y} - y) \odot H(S_4)\left(\widetilde{W}_4\right)^T] \odot (1 - a_3{}^2)a_2{}^T = \begin{bmatrix} \cdot & \cdot & \cdot \end{bmatrix}$$

$$\frac{\partial E}{\partial B_3} = \frac{\partial E}{\partial S_3} = \frac{\partial E}{\partial a_3} \odot \frac{\partial a_3}{S_3} = [(\hat{y} - y) \odot H(S_4)\left(\widetilde{W}_4\right)^T] \odot (1 - a_3{}^2) = \begin{bmatrix} \cdot \\ \cdot \end{bmatrix}$$

$$\frac{\partial E}{\partial \widetilde{W}_2} = \frac{\partial E}{\partial S_2}(a_1)^T = \frac{\partial E}{\partial a_2} \odot \frac{\partial a_2}{S_2}(a_1)^T = \left(\widetilde{W}_3\right)^T \frac{\partial E}{\partial S_3} \odot \frac{\partial a_2}{S_2}(a_1)^T$$

$$= [[\left(\widetilde{W}_3\right)^T [\left(\widetilde{W}_4\right)^T (\hat{y} - y) \odot H(S_4) \odot (1 - a_3{}^2)] \odot (a_2)(1 - a_2)](a_1)^T = \begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \end{bmatrix}$$

$$\frac{\partial E}{\partial B_2} = \frac{\partial E}{\partial S_2} = \left[\left(\widetilde{W}_3\right)^T \left[\left(\widetilde{W}_4\right)^T (\hat{y} - y) \odot H(S_4) \odot (1 - a_3{}^2)\right] \odot (a_2)(1 - a_2)\right] = \begin{bmatrix} \cdot \\ \cdot \end{bmatrix}$$

$$\frac{\partial E}{\partial \widetilde{W}_1} = \frac{\partial E}{\partial S_1}(X_1)^T = \frac{\partial E}{\partial a_1} \odot \frac{\partial a_1}{S_1}(X_1)^T = \left(\widetilde{W}_2\right)^T \left[\frac{\partial E}{\partial S_2} \odot (1 - a_1{}^2)\right](X_1)^T$$

$$= \left(\widetilde{W}_2\right)^T \left[\left[\left(\widetilde{W}_3\right)^T \left[\left(\widetilde{W}_4\right)^T (\hat{y} - y) \odot H(S_4) \odot (1 - a_3{}^2)\right] \odot (a_2)(1 - a_2)\right] \odot (1 - a_1{}^2)\right](X_1)^T$$

$$= \begin{bmatrix} \cdot & \cdot & \cdot \end{bmatrix}$$

$$\frac{\partial E}{\partial B_1} = (\widetilde{W}_2)^T\left[[(\widetilde{W}_3)^T[(\widetilde{W}_4)^T(\hat{y} - y) \odot H(S_4) \odot (1 - a_3{}^2)] \odot (a_2 - a_2{}^2)] \odot (1 - a_1{}^2)\right]$$

$$= \begin{bmatrix} \cdot \\ \cdot \end{bmatrix}$$

با توجه به صفر شدن تغییرات پارامترها، پارامترها ثابت مانده و دوباره مراحل فوق را برای ورودی X_2 تکرار می‌کنیم.

$$a_۱ = Z = \tanh\left(\widetilde{W}_۱ X_۲ + B_۱\right) = \tanh(S_۱) = \begin{bmatrix} ۰/۹۹۷۸ \\ ۰/۹۹۹۴ \end{bmatrix}$$

$$a_۲ = K = \text{sigmoid}\left(\widetilde{W}_۲ a_۱ + B_۲\right) = \text{sigmoid}(S_۲) \cong \begin{bmatrix} ۱ \\ ۱ \\ ۱ \end{bmatrix}$$

$$a_۳ = P = \tanh\left(\widetilde{W}_۳ a_۲ + B_۳\right) = \tanh(S_۳) \cong \begin{bmatrix} ۱ \\ ۱ \end{bmatrix}$$

$$a_۴ = \hat{y} = \text{ReLU}\left(\widetilde{W}_۴ a_۳ + B_۴\right) = \text{ReLU}(S_۴) \cong [۰]$$

$$E = \frac{۱}{۲}(\hat{y} - y_۱)^۲ = ۸$$

با توجه به صفر شدن $\hat{y}$، تغییرات برای پارامترها همانند قبل صفر شده و پارامترها به همان صورت اولیه باقی می‌مانند.

ب) شبیه سازی در پایتون

محاسبات بخش قبل را در پایتون به صورت مجموعه‌ای از توابع پیاده سازی کرده و خروجی را محاسبه می‌کنیم. مطابق انتظار خروجی حاصل با محاسبات بخش تحلیلی هم‌خوانی داشته و پارامترهای مسئله بدون تغییر باقی می‌ماند.

```
Weights:
W1 = [[0.12 0.32 0.52]
 [0.24 0.44 0.44]]

W2 = [[2.15 4.45]
 [2.25 4.55]
 [2.35 4.65]]

W3 = [[8.12 8.32 8.52]
 [8.22 8.42 8.62]]

W4 = [[-1.84 -1.64]]

Bias:
B1 = [[0.21]
 [0.42]]

B2 = [[6.15]
 [6.25]
 [6.35]]

B3 = [[0.52]
 [0.62]]

B4 = [[2.26]]
```

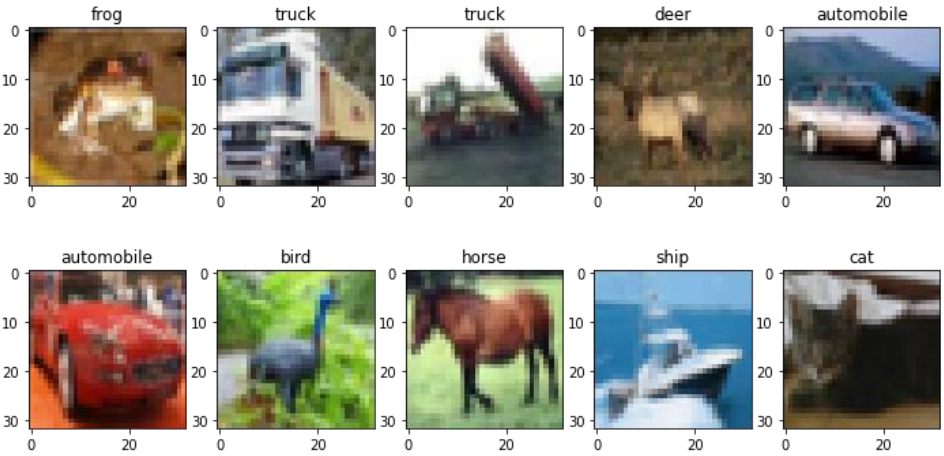
شکل ۱ـ۱: خروجی پارامترها پس از پیاده سازی شبکهٔ عصبی چند لایه در پایتون

سوال دوم) کاربرد شبکه‌های عصبی در طبقه بندی

در این بخش قصد داریم یک طبقه بند برای مجموعه دادهٔ $CIFAR-10$ با استفاده از شبکه‌های MLP و CNN آموزش داده و تاثیر هایپرپارامترها را بر این طبقه بند را بررسی می‌کنیم. این مجموعه داده شامل ۶۰ هزار تصویر رنگی میباشد که در ۱۰ کلاس دسته بندی شده است.

الف) استفاده از شبکه MLP

این مجموعه دادگان را دانلود کرده و ۱۰ تصویر ابتدایی را همراه با label آن‌ها نمایش می‌دهیم. سپس دادگان را به ۳ دستهٔ آموزش، تست و ارزیابی تقسیم می‌کنیم.



شکل ۱ـ۲: ۱۰ تصویر ابتدایی مجموعه دادگان CIFAR

سپس پیش‌پردازش‌های مورد نیاز را روی دادگان انجام داده و آن‌ها را آمادهٔ پردازش می‌کنیم. برای این سوال می‌توان از کتابخانهٔ Keras استفاده کرد. برای پیش پردازش دادگان، ابتدا داده‌ها را نرمالایز کرده و سپس تصاویر را به صورت بردار در آورده تا پردازش آن‌ها ساده تر شود. در نهایت نیز آن‌ها را به سه دستهٔ آموزش، ارزیابی و تست تقسیم می‌کنیم. (۱۰ درصد ابتدایی دادگان آموزش را به عنوان دادهٔ ارزیابی در نظر می‌گیریم.)

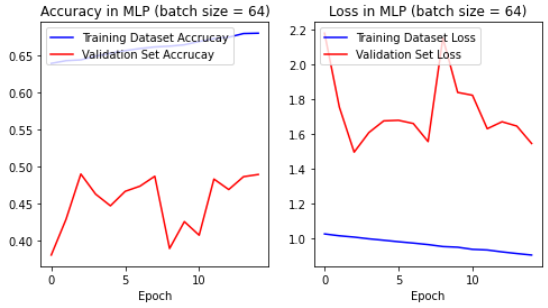
روش ابتدایی مورد استفاده در این بخش Stochastic mini batch based است. برای طراحی این شبکه تعداد لایه‌های مخفی را ۲ در نظر می‌گیریم. برای پیاده سازی این شبکه توابع فعالیت را ReLU در نظر می‌گیریم که ضابطهٔ آن به صورت زیر تعریف می‌شود:

$$RelU(x) = \max(0, x)$$

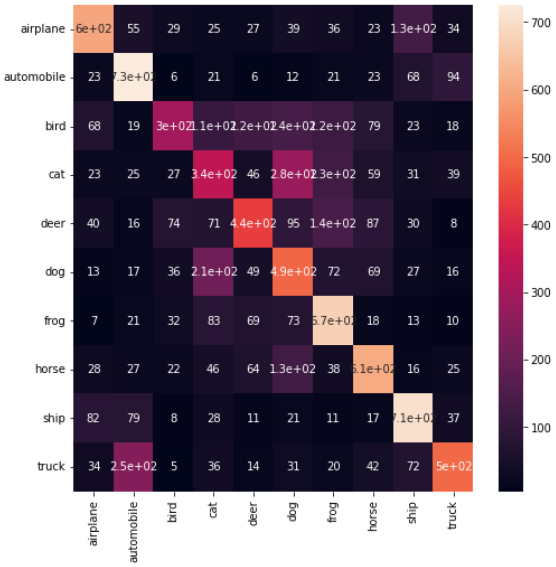
همچنین تابع هزینه را با استفاده از Cross Entropy تعریف می‌نماییم.

۱) از سه دسته با اندازه‌های ۳۲، ۶۴ و ۲۵۶ استفاده کرده و تاثیر تفاوت اندازهٔ دسته‌ها را در دقت و زمان آموزش شبکه بررسی می‌کنیم. دقت شود که تعداد epoch را برای آموزش ۱۵ در نظر می‌گیریم. می‌بینیم که هرچه قدر اندازه‌های دسته‌ها بیشتر شود، دقت روی داده‌های آموزش و ارزیابی بالاتر می‌رود چراکه در هر تکرار، مدل داده‌های بیشتری را می‌بیند و پارامترها را بهتر آپدیت می‌کند که باعث افزایش قدرت تعمیم‌پذیری مدل می‌شود. همچنین کارایی و سرعت آموزش نیز افزایش می‌یابد.

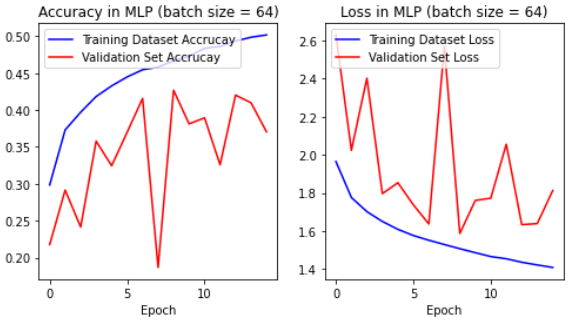
۲) توابع فعال ساز را سه مرتبه تغییر می‌دهیم. دقت شود که در حالت اولیه هردو تابع فعال‌ساز ReLU بوده است. در حالت دوم هردو تابع فعال ساز را SeLU ، در حالت سوم هر دو را sigmoid و درحالت چهارم نیز هردو را tanh قرار می‌دهیم. باتوجه به نمودارهای زیر و عملکرد مدل، می‌توان نتیجه گرفت که ReLU از بقیه بهتر عمل می‌کند. (دقت و خطای هر epoch در فایل کد، ضمیمه شده است)



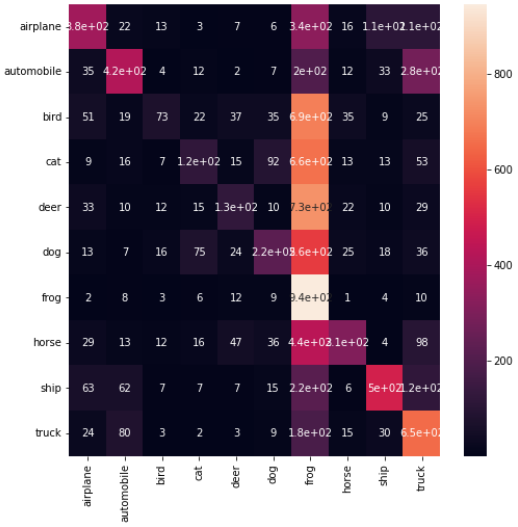
شکل ۲ـ۲: نمودار دقت و خطا در هر epoch با توابع فعال‌ساز ReLU



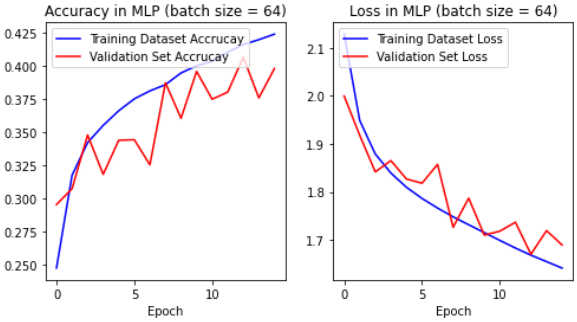
شکل ۳ـ۲: ماتریس آشفتگی روی داده‌های تست با توابع فعال‌ساز ReLU



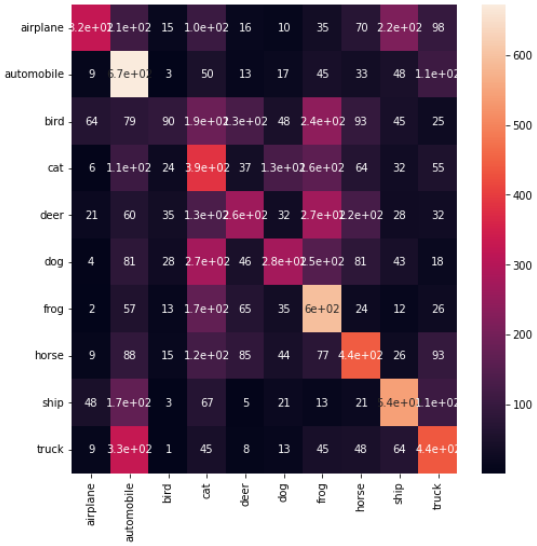
شکل ۴ـ۲: نمودار دقت و خطا در هر epoch با توابع فعال‌ساز SeLU



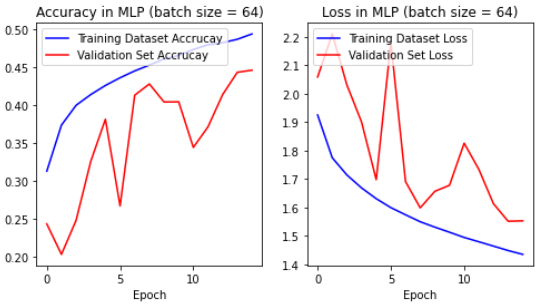
شکل ۵ـ۲: ماتریس آشفتگی روی داده‌های تست با توابع فعال‌ساز SeLU



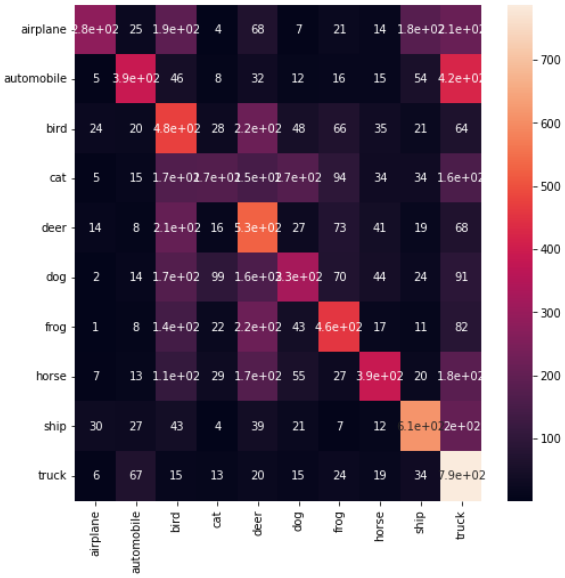
شکل ۶ـ۲: نمودار دقت و خطا در هر epoch با توابع فعال‌ساز sigmoid



شکل ۷ـ۲: ماتریس آشفتگی روی داده‌های تست با توابع فعال‌ساز sigmoid



شکل ۸ـ۲: نمودار دقت و خطا در هر epoch با توابع فعال‌ساز tanh



شکل ۹ـ۲: ماتریس آشفتگی روی داده‌های تست با توابع فعال‌ساز tanh

همانطور که پیش تر نیز اشاره شد، ReLU بهترین عملکرد را دارد؛ چراکه نورون‌ها در آن اشباع نشده و گرادیان kill نمی‌شود و مهم تر از آن بسیار سریع تر از دیگر روش‌ها همگرا می‌شود از مشکلات این روش نیز می‌توان به non zero-centered بودن آن اشاره کرد. در این روش خروجی منفی هرگز فعال نشده و آپدیت نمی‌شود.

تابع فعال‌ساز sigmoid سبب اشباع شدن نورون‌ها شده و گرادیان اصطلاحا kill می‌شود. همچنین خروجی این تابع فعال‌ساز zero-centered نمی‌باشد و گرادیان همواره یا مثبت است و یا منفی که این ویژگی‌ها نشان از efficient نبودن این روش است.

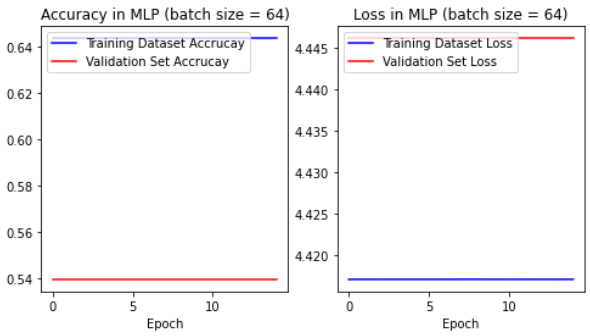
از مزایای تابع فعال ساز tanh می‌توان به این اشاره کرد که خروجی‌ای بین ۱- تا ۱ داشته و zero-centered می‌باشد اما بزرگترین مشکل آن این است که اگر نورون‌ها اشباع شوند گرادیان تغییر نمی‌کند.

ضابطهٔ SeLU به صورت زیر تعریف می‌شود:

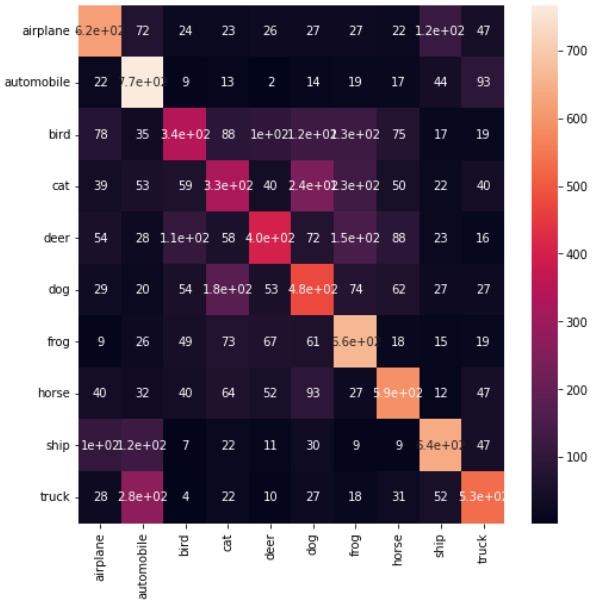
$$SeLU(x) = \begin{cases} x & x > 0 \\ \alpha e^x - \alpha & x \leq 0 \end{cases}$$

این روش شباهت زیادی به ReLU داشته با این تفاوت که برای ورودی‌های منفی، خروجی صفر نمی‌شود. اما همچنان ReLU، روشی کارا تر و سریع تر می‌باشد و معمولاً از ReLU به عنوان activation function استفاده می‌شود.

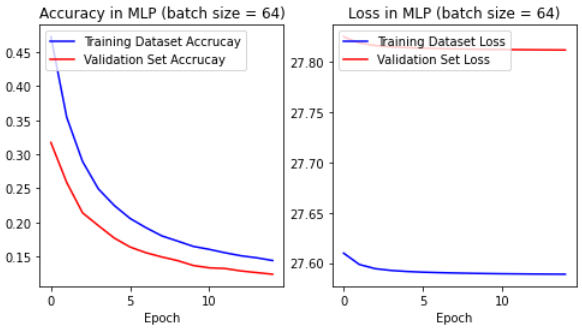
۳) اکنون تابع خطا را دو مرحله تغییر داده و نتایج را بررسی می‌کنیم. تابع خطای حالت اولیه را cross entropy، حالت دوم را mean absolute error و حالت سوم را mean squared error قرار می‌دهیم. با اجرای مدل، می‌بینیم که دقت زمانی که loss function را cross entropy قرار می‌دهیم، دقت روی داده‌های validation و train از بقیهٔ حالات بیشتر است.



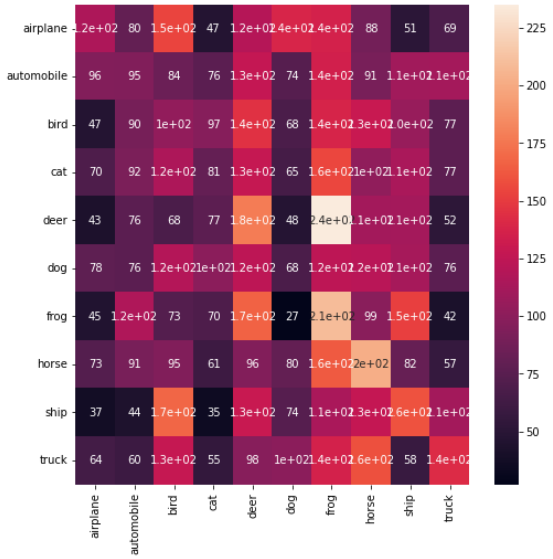
شکل ۱۰ـ۲: نمودار دقت و خطا در هر epoch با تابع خطای mean absolute error



شکل ۱۱ـ۲: ماتریس آشفتگی روی داده‌های تست با تابع خطای
mean absolute error



شکل ۱۲ـ۲: نمودار دقت و خطا در هر epoch با تابع خطای
mean squared error



شکل ۱۳ـ۲: ماتریس آشفتگی روی داده‌های تست با تابع خطای
mean squared error

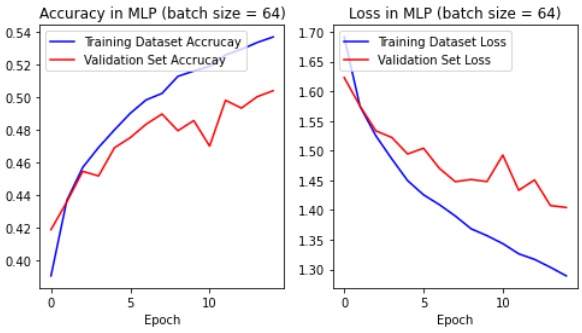
معیار cross entropy به دلیل بهره گیری از معیارهای تئوری اطلاعات، قابلیت بهتری برای ارزیابی عملکرد مدل و محاسبهٔ خطا دارد. معیار MSE، معمولاً در رگرسیون و cross entropy در classification به کار می‌رود چرا که در طبقه بندی ما با مجموعه خاصی از مقادیر خروجی کار می‌کنیم. به عنوان مثال در نظر می‌گیریم که target به صورت [۱,۰,۰,۰] و predicted output پس از اعمال softmax به صورت زیر می‌باشد:

[۰/۶,۰/۴,۰,۰]

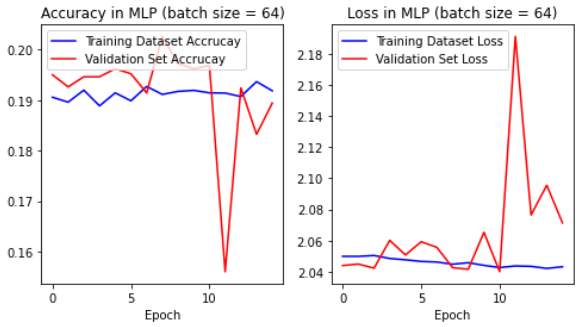
در این حالت cross entropy loss برابر است با ۰/۷۴ و MSE loss نیز برابر است با ۰/۰۸.

حال اگر predicted output به صورت [۰/۴,۰/۶,۰,۰] باشد، cross entropy loss برابر با ۱/۳۲ و MSE loss برابر با ۰/۱۲ می‌شود. همانطور که می‌بینیم MSE یا MAE در این حالت توانایی خوبی در تمییز دادن این دو حالت ندارند و cross entropy کارایی بهتری نسبت به این دو در تشخیص بهتر خطا خواهد داشت.

۴) در این بخش، توابع بهینه ساز را دو مرحله تغییر می‌دهیم. حالت اولیه SGD، حالت بعدی Adam و حالت سوم را RMSprop در نظر می‌گیریم. پس از اجرای مدل‌ها، نمودارهای دقت و خطا در هر eopch را رسم می‌کنیم.



شکل ۱۴ـ۲: نمودار دقت و خطا در هر epoch با ADAM Optimizer



شکل ۱۵ـ۲: نمودار دقت و خطا در هر epoch با RMSprop Optimizer

با مقایسهٔ این حالات می‌بینیم که بهترین تابع بهینه ساز SGD بوده که دقتی در حدود ۵۳ درصد برای زمانی که batch size برابر با ۶۴ است دارد. به طور کلی می‌توان گفت تابع بهینه ساز Adam، سریع تر باعث

همگرایی شده اما قدرت تعمیم پذیری کمتری نسبت به SGD دارد. الگوریتم SGD به این صورت است که به جای محاسبهٔ گرادیان روی تمام داده‌ها و دیدن تمام آن‌ها، با انتخاب یک random selection از دیتا در هر تکرار، گرادیان را آپدیت می‌کند. الگوریتم RMSprop نیز با میانگین مجذور گرادیان‌ها را محاسبه کرده و گرادیان را بر جذر این مقدار تقسیم کرده و سپس پارامترها را آپدیت می‌کند. روش بهینه سازی Adam، یک روش گرادیان نزولی تصادفی مبتنی بر تخمین تطبیقی momentum های مرتبهٔ اول و دوم است. Adam معیار مناسبی برای مسائل بزرگ که دارای تعداد زیادی دیتا و پارامتر هستند، می‌باشد.

۵) با توجه به ارزیابی‌های انجام شده، بهترین مدل، مدلی با تابع بهینه ساز SGD، تابع خطای cross entropy و با activation functionهای ReLU با $batch\ size = ۲۵۶$ است.

برای این مدل معیارهای ارزیابی Precision، Recall و F-Score را به دست می‌آوریم.

```
precision_recall_fscore_support(y_test[:,0], pred_labels, average=None)
```

```
(array([0.66118837, 0.62283105, 0.44191097, 0.3434238 , 0.50477327,
        0.41493384, 0.53658537, 0.67185473, 0.55636364, 0.57632399]),
 array([0.523, 0.682, 0.407, 0.329, 0.423, 0.439, 0.66 , 0.518, 0.765,
        0.555]),
 array([0.58403127, 0.65107399, 0.42373764, 0.3360572 , 0.46028292,
        0.42662779, 0.59192825, 0.58498024, 0.64421053, 0.56546103]),
 array([1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000]))
```

شکل ۱۶ـ۲: محاسبهٔ معیارهای ارزیابی F-Score، Recall و Precision

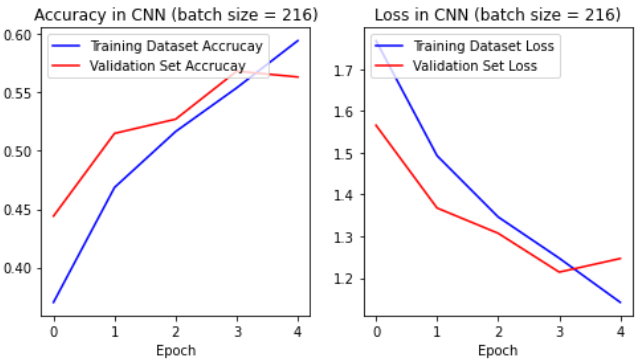
ب) استفاده از شبکه MLP و CNN

در این بخش قصد داریم عملکرد شبکهٔ MLP را با استفاده از لایه‌های کانوولوشنی بهبود بخشیم.

۱) در ابتدا لایه‌های کانوولوشنی را به وسیلهٔ تابع $Conv۲D$ در پایتون اضافه کرده و مدل را ارزیابی می‌کنیم.

```
Model: "Sequential_CNN1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_8 (Conv2D)           (None, 32, 32, 32)        416

 conv2d_9 (Conv2D)           (None, 32, 32, 32)        4128

 flatten_4 (Flatten)         (None, 32768)             0

 dense_12 (Dense)            (None, 300)               9830700

 dense_13 (Dense)            (None, 100)               30100

 dense_14 (Dense)            (None, 10)                1010

=================================================================
Total params: 9,866,354
Trainable params: 9,866,354
Non-trainable params: 0
_________________________________________________________________
```

شکل ۱۷ـ۲: CNN Model Summary



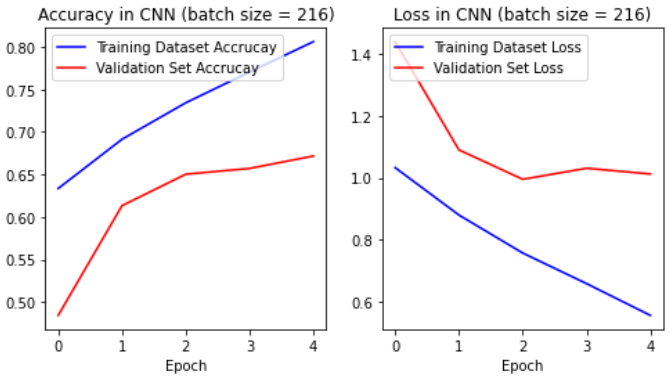
شکل ۱۸ـ۲: نمودار دقت و خطا در هر epoch در شبکهٔ کانوولوشنی اولیه

در نهایت نیز مدل را روی دادگان تست با استفاده از model.evaluate ارزیابی می‌کنیم که با توجه به خروجی حاصل مقدار تابع هزینه برابر با ۱/۲۷ و مقدار دقت نیز حدود ۵۵ درصد می‌باشد که به نسبت شبکهٔ MLP، مقدار بهتری است.

۲) یکی از مشکلات در شبکهٔ عصبی این است که خروجی ممکن است یا خیلی کم و یا خیلی بزرگ باشد که باعث مختل کردن فرایند آموزش در لایه‌های بعدی شود. batch normalization سبب مستقل شدن آموزش داده‌ها و مناسب کردن توزیع خروجی‌ها می‌شود.

Max pooling یک pooling operation است که برجسته‌ترین ویژگی‌های مدل را حفظ می‌کند.

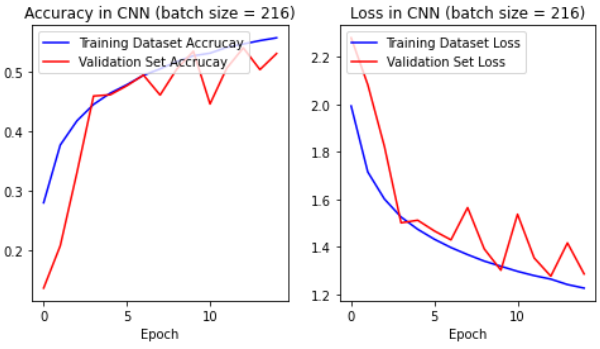
اکنون لایه‌ها را اضافه کرده و نتایج را بررسی می‌نماییم.



شکل ۱۹ـ۲: نمودار دقت و خطا در هر epoch در شبکهٔ کانوولوشنی با Max pooling و Batch normalization

دقت و خطا نیز روی داده‌های تست به ترتیب حدود ۶۵ درصد و ۱/۰۸ به دست می‌آید که نشان می‌دهد مدل عملکرد بهتری نسبت به MLP و CNN بدون batch normalization و max pooling است.

۳) در این مرحله برای بهبود عملکرد شبکهٔ عصبی، از Dropout استفاده می‌نماییم. Dropout به این گونه عمل می‌کند که بعضی نورون‌ها را به صورت رندوم، خاموش کرده و از بقیهٔ نورون‌ها برای آپدیت کردن وزن‌ها کمک می‌گیرد.

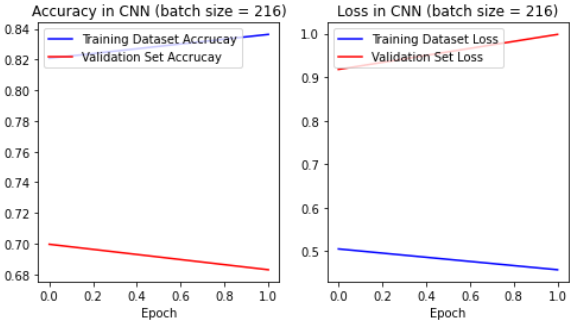


شکل ۲۰ـ۲: نمودار دقت و خطا در هر epoch در شبکهٔ کانوولوشنی با Max pooling، Batch normalization و Dropout

همچنین دقت و خطا روی داده‌های تست به ترتیب حدود ۰/۶۷ و ۶۷ درصد می‌باشد.

۴) در Early Stopping، باید خطا را برحسب تعداد epoch رصد کنیم. معمولاً برای دادهٔ آموزش روند تغییر خطا، کاهشی بوده و برای داده‌های ارزیابی نیز ابتدا کاهشی و سپس افزایشی است. Early Stopping در بهترین نقطه، توقف کرده و وزن را در آن نقطه به عنوان خروجی در نظر می‌گیرد. چراکه در ابتدای یادگیری، سطوح تصمیم گیری به صورت کلی ایجاد شده و هرچه قدر که الگوریتم جلوتر می‌رود، سطوح تصمیم‌گیری

جزئی تر می‌شود. برای اضافه کردن Early Stopping نیازی به اضافه کردن لایه نیست و با استفاده از کتابخانهٔ keras این معیار را اعمال می‌نماییم.



شکل ۲۱ـ۲: نمودار دقت و خطا در هر epoch در شبکهٔ کانوولوشنی با Max pooling، Batch normalization و Dropout با اعمال توقف زودهنگام

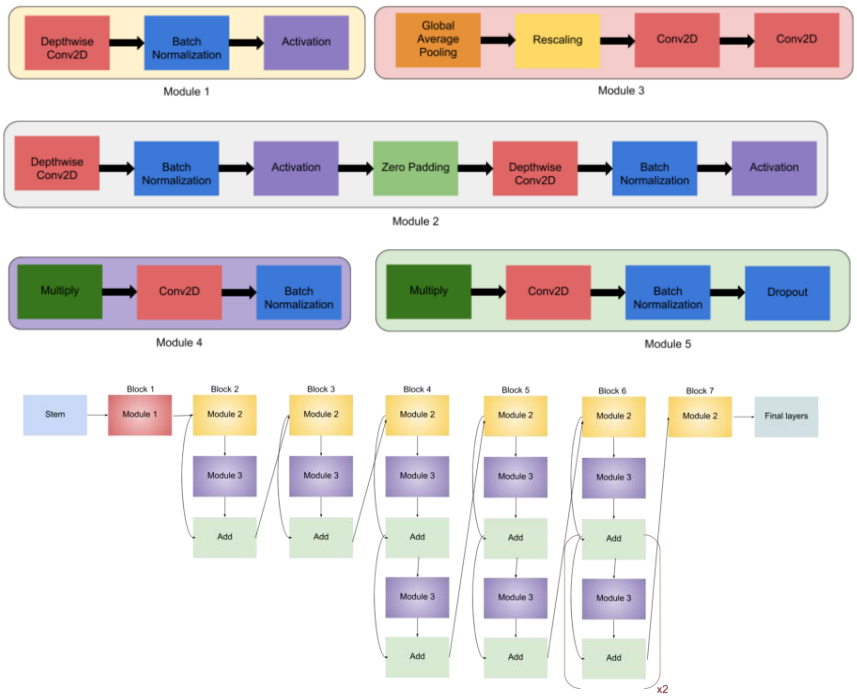
دقت و خطا در این حالت بر روی دادگان تست به ترتیب تقریباً برابر است با ۶۷ درصد و ۱/۰۳ مطابق انتظار نتیجه می‌گیریم که نسبت به حالت قبل عملکرد مدل بهتر شده است.

سوال سوم) یادگیری انتقال‌یافته برای شبکهٔ Efficient Net

الف) آشنایی با شبکهٔ Efficient Net

۱) معماری شبکه:

این شبکه، یک شبکهٔ کانوولوشنی با معماری‌ای مطابق شکل زیر است. در این شبکه از ماژول‌های ۱ تا ۳ استفاده شده است که لایه‌های استفاده شده در هر ماژول در شکل زیر نمایش داده شده است. دلیل نام‌گذاری این شبکه نیز این است که با تعداد معقولی پارامتر، عملکرد بسیار خوبی را در تشخیص و classification ارائه می‌کند.



شکل ۱ـ۳: شبکهٔ Efficient Net ($B0$)

۲) نسخه‌های مختلف معماری و تفاوت آن‌ها:

شبکهٔ Efficient Net نسخه‌های متفاوتی با نام‌های $B1$ تا $B7$ دارد که تفاوت این نسخه‌ها در ماژول‌های به کار برده شده در شبکه و ترتیب قرار گیری آن‌های در هر بلاک دارد. به طور کلی از $B1$ تا $B7$، تعداد پارامترها و به تبع آن، عمق شبکه افزایش پیدا می‌کند. تمام این شبکه‌ها بر پایهٔ EfficientNet $B0$ توسط مهندسین طراحی شده‌اند.

۳) پیش پردازش اولیه برای تصویر ورودی:

باید دقت کرد که ورودی این شبکه باید تصویری با ابعاد (۱,۲۲۴,۲۲۴,۳) باشد. بنابراین قبل از compile کردن شبکه روی داده‌های مد نظر، ابعاد دادگان را مطابق ابعاد قابل قبول این شبکه، تغییر دهیم.

۴) مزایا نسبت به سایر مدل‌ها:

در شبکه‌های کانوولوشنی برای رسیدن به دقت بیشتر، نیاز به پارامترهای بیشتری داریم که افزایش تعداد پارامترها، ما را با محدودیت حافظه رو به رو می‌کند. بنابراین به دنبال این محدودیت، شبکه‌های EfficientNet طراحی شده که مدل‌های کارآمدتری برای رسیدن به دقت بیشتر هستند. در این شبکه‌ها، عمق و رزولوشن به صورت اصولی و به تدریج افزایش می‌یابد.

ب) پیاده سازی شبکه به کمک ایدهٔ Transfer Learning

شبکهٔ $Efficient\ NetB0$ را در پایتون به وسیلهٔ کتابخانهٔ keras پیاده سازی کرده و عکس زیر را به عنوان ورودی تابع به آن می‌دهیم. دقت شود که ورودی این شبکه، باید ابعادی به صورت (۱,۲۲۴,۲۲۴,۳) داشته باشد. بنابراین باید تصویر را reshape کرده و سپس کلاس آن را مشخص کنیم.

شکل ۲ـ۳: تصویر گرفته شده از محیط اطراف

```
Detected Object : rocking_chair, Probability: 24.9%
Detected Object : barrow, Probability: 7.75%
Detected Object : swing, Probability: 3.19%
```

شکل ۳ـ۳: سه دستهٔ تشخیص داده شده با بیشترین احتمال

با توجه به خروجی بالا، عکس گرفته شده، با احتمال تقریبی ۲۵ درصد صندلی سنگی تشخیص داده شده است.

ج) رفع یک مشکل خاص شبکه

برای حل این مشکل می‌توان دسته با بیشترین احتمال را مورد بررسی قرار داد و اگر احتمال آن از آستانهٔ مشخص شده (به عنوان مثال ۲۰ درصد) کمتر بود، دستهٔ تشخیص داده شده را رد کنیم.

```
def prediction(img,condition_check):
  pred = effnetB0.predict(img)
  if np.max(pred)*100<20 and condition_check:
    print('The Network could not classify the object.')
    print('Maybe the object does not exist in the list of the labels!')
  else:
    for i in range(3):
      name = decode_predictions(pred)[0][i][1]
      prob = decode_predictions(pred)[0][i][2]*100
      print(f'Detected Object : {name}, Probability: {np.round(prob, 2)}%')
```

شکل ۴ـ۳: تابع نوشته شده جهت رد کردن تصویر با احتمال تشخیص کمتر از ۲۰ درصد

برای آزمودن تابع فوق، عکسی از یک کیف کولی را گرفته و در colab، بارگذاری می‌کنیم و آن را به عنوان ورودی به تابع می‌دهیم. همانطور که در تصویر زیر می‌بینیم شی تشخیص داده شده توسط تابع، رد شده است.

```
The Network could not classify the object.
Maybe the object does not exist in the list of the labels!
```

شکل ۵ـ۳: تصویر داده شده به تابع پیش‌بینی و رد شدن تشخیص تابع

د) آموزش شبکه با مجموعه دادگان جدید

با توجه به لیست اشیا قابل تشخیص توسط این شبکه، یک مجموعه داده جمع آوری کرده که شامل دو دسته باشد و مدل را با استفاده از این مجموعه داده مجدداً آموزش می‌دهیم.

برای این کار ابتدا دو مجموعهٔ داده را انتخاب می‌کنیم. به عنوان مثال موتور سیکلت و کامپیوتر. حال ۲۰۰ عکس برای هرکدام از این مجموعه‌ها پیدا کرده و هرکدام را به صورت فولدر در Google Colab آپلود می‌نماییم. سپس با نوشتن یک تابع این عکس‌ها را به عنوان دیتای آموزش در پایتون import کرده و با انجام پیش‌پردازش‌های لازم، شبکهٔ مد نظر را با اضافه کردن یک لایه به شبکهٔ EfficeintNet با دستور Dense و با تابع فعال‌ساز Softmax، آموزش می‌دهیم.

برای آزمودن این مدل، یک دسته دادهٔ ارزیابی نیز جمع آوری کرده و مدل را براساس این داده‌ها ارزیابی می‌نماییم.

در نهایت مدل را compile کرده و مقدار loss و accuracy را در هر epoch مطابق نتایج زیر به دست می‌آوریم. همانطور که در نتایج زیر مشاهده می‌شود، دقت این مدل روی داده‌های validation بسیار بالا بوده که نشان از کارایی بالای این شبکهٔ عصبی دارد.

```
142s 10s/step - loss: 0.6906 - accuracy: 0.8225 - val_loss: 0.6816 - val_accuracy: 0.7750 - lr: 1.0000e-04
123s 9s/step - loss: 0.6843 - accuracy: 0.9700 - val_loss: 0.6766 - val_accuracy: 0.8250 - lr: 1.0000e-04
124s 10s/step - loss: 0.6748 - accuracy: 0.9825 - val_loss: 0.6650 - val_accuracy: 0.9500 - lr: 1.0000e-04
122s 9s/step - loss: 0.6609 - accuracy: 0.9950 - val_loss: 0.6442 - val_accuracy: 1.0000 - lr: 1.0000e-04
125s 10s/step - loss: 0.6479 - accuracy: 0.9925 - val_loss: 0.6290 - val_accuracy: 1.0000 - lr: 1.0000e-04
```

شکل ۶ـ۳: مقادیر دقت و تابع هزینه برای دادگان آموزش و ارزیابی در هر epoch