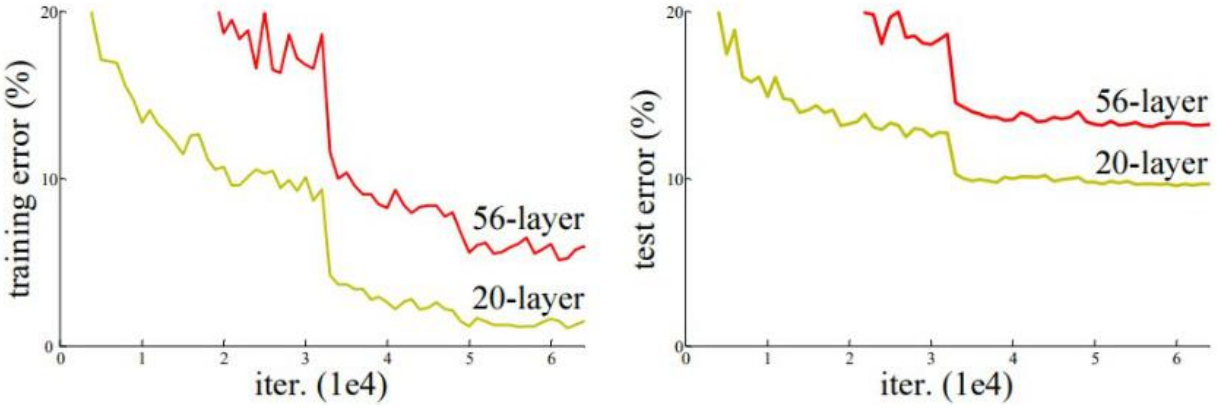
تشخیص حروف الفبای انگلیسی به وسیله شبکه های عصبی کانولوشنی و رزنت

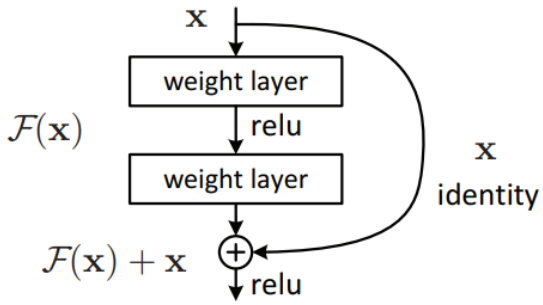
معماری شبکه عصبی ResNet

ResNet که در سال 2015 توسط محققان مایکروسافت ریسرچ پیشنهاد شد، معماری جدیدی به نام Residual Network را معرفی کرد. در معماری های شبکه عصبی هرچه لایه های بیشتری در یک شبکه عصبی عمیق استفاده می کند میزان خطا را کاهش دهد. این برای تعداد لایه‌های کمتری کار می‌کند، اما وقتی تعداد لایه‌ها را افزایش می‌دهیم، یک مشکل رایج در یادگیری عمیق مرتبط با آن وجود دارد که گرادیان Vanishing/Exploding نامیده می‌شود. این باعث می شود که گرادیان 0 یا خیلی بزرگ شود. بنابراین وقتی تعداد لایه ها را افزایش می دهیم، میزان خطای آموزش و آزمون نیز افزایش می یابد.



در نمودار بالا، می‌توانیم مشاهده کنیم که یک CNN 56 لایه نسبت به معماری 20 لایه CNN، میزان خطای بیشتری را در مجموعه داده‌های آموزشی و آزمایشی ارائه می‌دهد. پس از تجزیه و تحلیل بیشتر در مورد میزان خطا، نویسندگان توانستند به این نتیجه برسند که علت آن vanishing/exploding gradient است .

به منظور حل مشکل این معماری مفهومی به نام Residual Blocks را معرفی کرد . در این شبکه از تکنیکی به نام skip connections استفاده میکنیم . که لایه ها را با این تکنیک به لایه بعدی متصل میکنند و Resnet ها با چیدن این بلوک های باقیمانده در کنار هم ساخته می شوند .



مزیت افزودن این نوع اتصال پرش این است که اگر هر لایه ای به عملکرد معماری لطمه بزند، با تنظیم کردن از آن عبور می کند.

درصورتیکه نگاشت همانی، نگاشت بهینه مورد نظر باشه، نزدیک کردن تابع residual به صفر خیلی ساده تر از بدست آوردن نگاشت همانی توسط چندین لایه کانولوشنی پشت سر هم با توابع غیرخطی هست . در واقع اینطور میتوان بیان کرد که شبکه هرجا لازم بدونه لایه های وزن دار رو در نظر میگیره و اثر اون رو تو خروجی میاره و هرجا نیاز به نگاشت همانی بود، ضرایب کانولوشنی رو به سمتی میبره که مثل نگاشت همانی عمل کنه . گرادیان از هر دو مسیر عبور میکنه از skip ها عبور میکنه تا گرادیان به لایه های ابتدایی هم برسه تا وزن بهتر و دقیق تر آپدیت بشن و کارایی شبکه بالا بره از لایه ها هم باید عبور کنه تا هر جا لازم بود ضرایب رو جوری تنظیم و آپدیت کنه که تابع residual رو صفر کنن هرجا هم لازم نبود کار خودشو میکنه و اثرش میاد تو خروجی در هر حال اون وزن ها آپدیت میشوند .

فرمول نگاشت به صورت زیر است

`F(x) := H(x) - x which gives H(x) := F(x) + x.`

Architecture

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

در بحث BP این قاعده که خطا و ارور از انتها برمیگرده به ابتدا و همه لایه ها را آپدیت میکنه وجود دارد حال این خطا که بر میگرده از تمام مسیر ها شبکه به لایه قبلی برمیگرده . وقتی skip conection رو قرار میدهیم زمانی که گرادیان یکی از وزن ها صفر شد از مسیر آن گرادیان حرکت نمیکنه و با skip connection عبور میکنه از لایه ای که وزن آن صفر شده است . وقتی skip connection نبود آن لایه نمیتوانست نقش همانی پیدا کند وتا اگر هم وزنی ندارد ورودی و خروجی یکی شوند .

مورد بعد راجع به ساختار لایه های کانولوشنی اینکه همانطور که در شکل بالا میبینیم ما کانولوشن های 1*1 را در لایه قرار میدهیم و این باعث میشود که تعداد فیچرهای در اون لایه کم یا زیاد بکنیم مثلا 64 تا فیچر داریم و میرسونیم به 128 فیچر البته آن عدد 128 نشان دهنده 128 نوع مختلف فیلترها هستند . در واقع تک تک صفحه هات ورودی را در یک وزن ضرب کرده و این باعث افزایش فیچر میشود و سایز ورودی را کاهش نمیدهد .

ساختار دیتاست

دیتا ست ما شامل سه فایل train , validation, test میباشد که این تصاویر شامل حروف الفبای انگلیسی که به صورت حرکات دست برای ناشنوایان نشان میدهد مباشد . که ما این تصاویر و در گوگل درایو آپلود کرده و سپس با اتصال کولب به آن به دیتا دسترسی پیدا میکنیم .

پیاده سازی معماری Resnet-50

الف) با استفاده از ImageDataGenerator برای تولید دسته ای از داده های تصویر بصورت عدد که برای آموزش یک مدل یادگیری عمیق شروع به دسته بندی داده ها میکنیم .

کد چندین پارامتر را برای تقویت داده ها مشخص می کند:

1- preprocessing_function=preprocess_input نشان می دهد که یک تابع پیش پردازش به نام preprocess_input برای هر تصویر اعمال می شود.
2- shear_range=0.2 محدوده ای را مشخص می کند که در آن تبدیل های برشی تصادفی می توانند روی تصاویر اعمال شوند.
3- horizontal_flip=True نشان می دهد که چرخش های افقی تصادفی را می توان روی تصاویر اعمال کرد.
4- validation_split=0.4 مشخص می کند که 40 درصد از داده ها برای اعتبارسنجی استفاده می شود، در حالی که 60 درصد باقی مانده برای آموزش استفاده می شود.

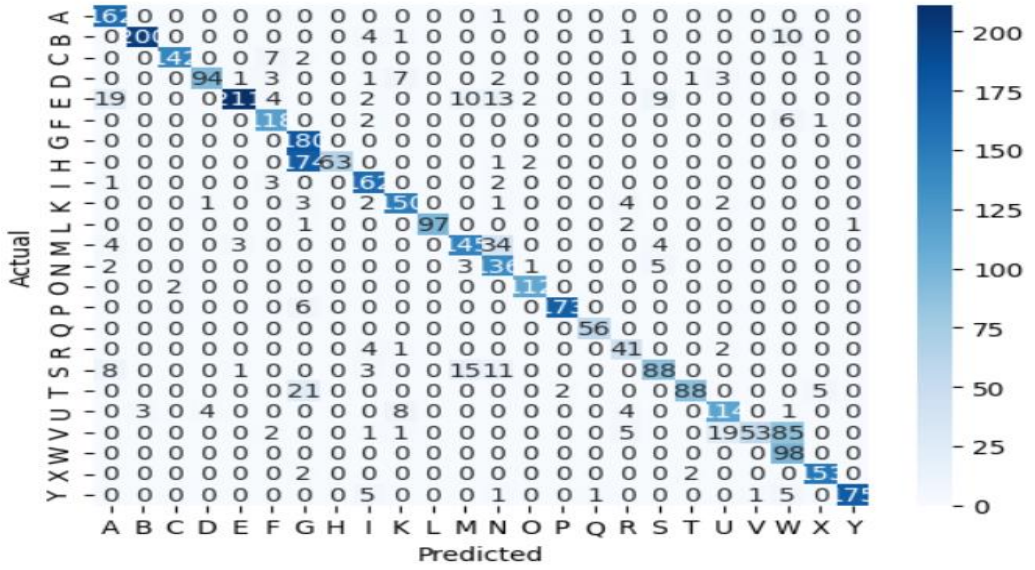
ب) با استفاده از تابع flow_from_directory از شی train_datagen یک generator برای داده های آموزشی ایجاد می کند که :

1- train_data_dir: این مسیر دایرکتوری است که داده های آموزشی در آن قرار دارند.
2- target_size: اندازه مورد نظر تصاویر را در داده های آموزشی مشخص می کند. چند برابر (img_width، img_height) طول می کشد.
3- batch_size: تعداد نمونه های هر دسته را در طول آموزش مشخص می کند.
4- class_mode: این نوع برچسب هایی که باید تولید شوند را مشخص می کند. در این مورد، روی "categorical" تنظیم می شود

به همین ترتیب test_datagen و validation_datagen را محاسبه میکنیم . پس از ان یک لایه resnet 50 را از API Keras دریافت کرده و بعد یک MaxPooling برای کاهش ابعاد تصاویر اعمال میکنیم و بعد یک لایه mlp با اکتیویشن فانکشن relu و بعد از آن softmax اعمال خواهیم کرد .

دقت مدل و confusion matrix

مقدار دقت مدل برابر است با 0.83 درصد و مقدار دقت تست مدل برابر با 0.833333 درصد میباشد که البته با تنظیم مقدار epoch مناسب میتوان میتوان به دقت مناسبی دست یافت . همچنین میتوان confusion matrix را برای مدل بالا محاسبه نمود که برابر مقدار زیر است .



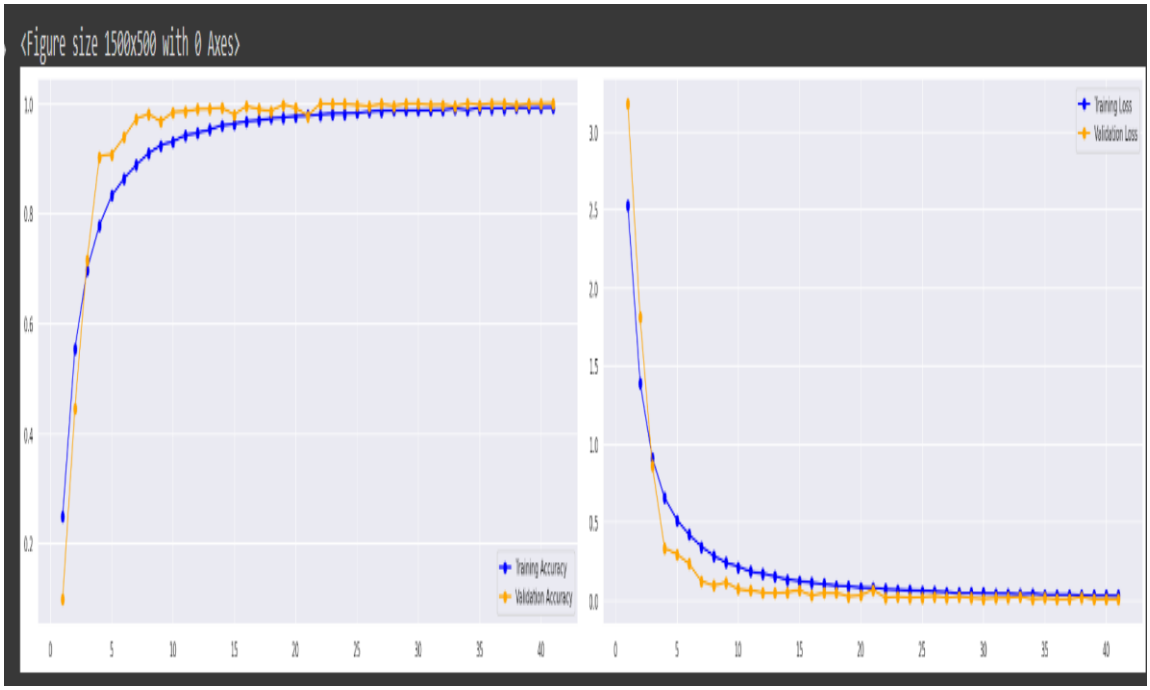
پیاده سازی با استفاده از CNN

طبق روال قبلی چون داده هامون عکس هستند ابتدا با استفاده از تابع ImageDataGenerator از API Kera عکسامون و به عدد تبدیل میکنیم . سپس یک لایه کانولوشن با اکتیویشن فانکشن relu و یک لایه BatchNormalization برای نرمال سازی خروجی های بدست آمده استفاده میکنیم و در آخر با MaxPool2D ابعاد لایه رو کاهش میدهیم و البه به تعداد لایه ها اضافه میشود . پس از فلت کردن تصاویر با اعمال یک لایه dropout و بعد یک لایه mlp دیگر به ساختارمناسبی میرسم .

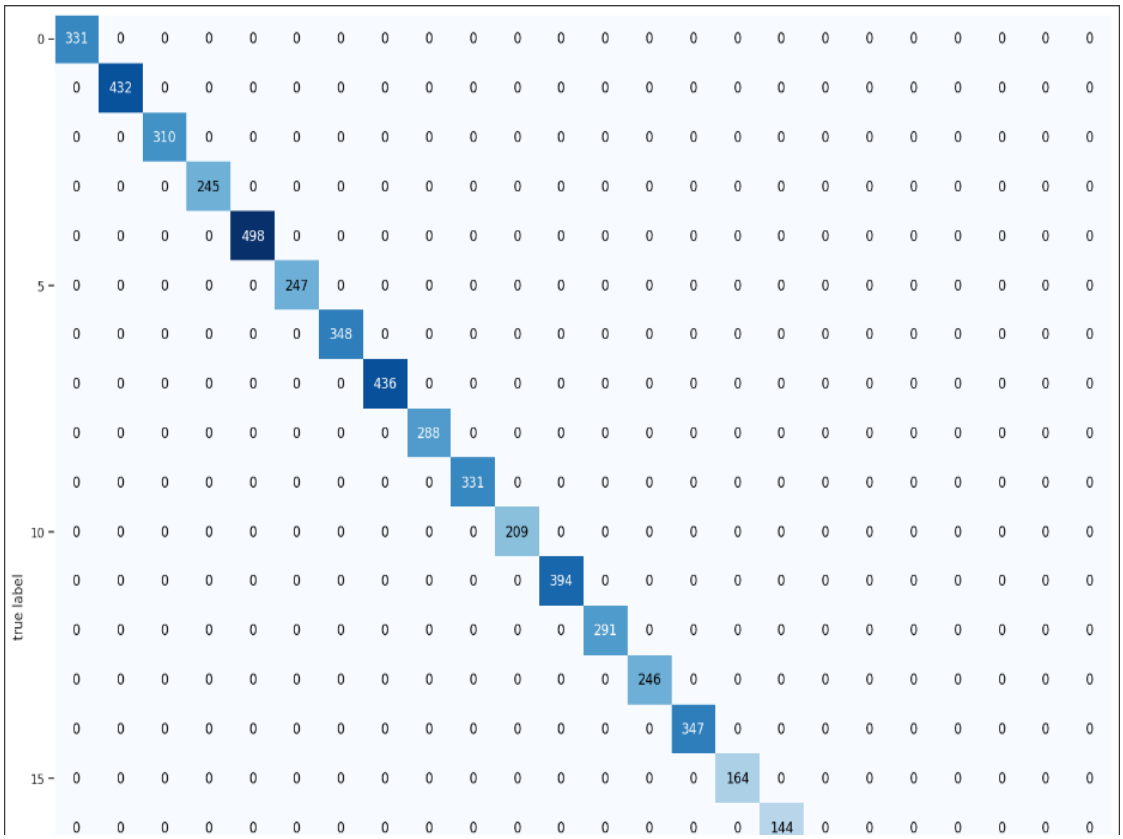
برای تشریح بیشتر اشاره ای به لایه dropout(0.2) میکنیم . این لایه حین آموزشِ این نورون‌ها، از تعدادی از آن‌ها به صورت تصادفی چشم‌پوشی شود. چشم‌پوشی یعنی اینکه آن نورون‌های خاص، در مسیر رفت یا برگشت در نظر گرفته نمی‌شوند.که با قرار دادن مقدار 0.2 نشان میدهد در طول فرآیند آموزش مدل مقدار 20 درصد از آن در نظر گرفته نمیشود .

دقت مدل

دقت مدل 0.99 درصد و دقت تست آن 0.9998 میباشد و نمودار در زیر نشاندهنده تغیرات دقت و مقدار loss را مشاهده میکنید .



Confusion matrix



به طور کلی برای مقایسه الگوریتم ها

1- مقدار ران تایم الگوریتم CNN نسبت به ResNet کمتر بوده .

2- دقت الگوریتم ResNet و CNN باهم برابر هستند و قابل تنظیم اند به وسیله callback_function و تغییرات مقدار epoch ها .

3- پیچیدگی زمانی الگوریتم CNNکمتر است .

پایان