به نام خدا

پوشه cnn شامل ۳تا mat. است که اسامی آنها عبارتند از startlearning وdemoMLCNN و .mlcnn

فایل startlearning برای اضافه کردن مسیر فایل های اجرا شده به direction متلب انجام می شود.

سپس تنظیمات را در config.mat. ذخیره می کنیم.

درون پوشه data داده های تصاویر به صورت پترن قرار دارند.پوشه visualizasion و utils برای بصری نشان دادن عملکرد لایه ها وخروجی است.

قسمت اصلی برنامه در فایل های demoMLCNN و mlcnn قرار دارند.برنامه نویسی به صورت شی گرایی بوده است.که شی ما در واقع mlcnn است.با استفاده از دستورclassdef یک ساختار ایجاد می کنیم و درون این ساختار به ترتیب پارامتر ها و توابع مورد نیاز برای آموزش شبکه را تعریف می کنیم.داخل این شی همه توابع feedforward و backpropagation وupdate وزن ها قرار دارند.

داخل mlcnn پارامتر های اولیه را set می کنیم.

ابتدا با دستور load داده ها را لود می کنیم.پترن ما یک ماتریس ۴۰۰۰۰*۴۰۰ خواهد بود.که ۴۰۰تعداد نمونه های ما است و هر نمونه ۴۰۰۰۰ بعد دارد.از آنجایی که ورودی CNN باید به صورت دوبعدی باشد ،یعنی مانند تصویر باشد،پترن ۴۰۰۰۰بعدی را به صورت دوبعدی ۲۰۰*۲۰۰ تعریف می کنیم.

چون تصویر ما سیاه وسفید است،عمق داده های ما ۱ خواهد بود ولی اگر مثلا رنگی بود داده های ما مثل یک مکعب به نرون های ورودی اعمال می شدند.

سپس وارد قسمت demoMLCNN می شویم.اینجا قسمتی است که شکل و ساختار اصلی شبکه کانوشن تعیین می گردد.شبکه دارای یک لایه ورودی دو لایه کانولش و دو لایه پولینگ است که ترتیب قرار گیری به این صورت است که بعد از هر لایه کانوشن یک لایه پولینگ قرار می گیرد.

بعد از آخرین لایه پولینگ،یک لایه مخفی داریم و بعد لایه خروجی قرار دارد.ساختار و تعداد نرون های هر لایه در کد های زیر مشخص شده است:

```
arch = {struct('type','input','dataSize',dataSize), ...
```

```
        struct('type','conv','filterSize',[5 5], 'nFM', 6), ...

        struct('type','subsample','stride',[2 2]), ...

        struct('type','conv','lRate',0.01,'filterSize',[4 5],

'nFM',12,'actFun','tanh'), ...
        struct('type','subsample','stride',2), ...

struct('type','hidden','lRate',0.000001,'nFM',50,'actFun','tanh'
),...
        struct('type','output', 'nOut', 2)};
```

ساختار شبکه ما به صورت یک آرایه سلولی متشکل از ساختار ها تشکیل شده است.ویژیگی های هر لایه درون سلول مخصوص خود توسط یک ساختار بیان شده است.هر آرایه سلول نشان دهنده ی لایه مورد نظر ما است.

Datasize متغیری است که هرسری ورودی درون آن قرار میگیرد.

منظور از filtersize سایز فیلتری است که با آن عمل کانولوشن n بعدی با دستورconvn درون تابع self.train داریم.

در ادامه به توضیح هر یک می پردازیم.

سایز فیلتر کانوشن اول [5 5] و دومی [4 5] است.تعداد نرون های کانولوشن اولی ۶ و دومی ۱۲ است. تابع فعال سازی اولی همانی و دومی tanh است.

در هر دو لایه پولینگ downsampling با ماتریس مربعی ۲*۲ انجام میشود.

تابع فعال سازی لایه hidden، tanh است و تعداد ۵۰ نرون دارد.

تعداد نرون های خروجی ۲ است.

نرخ یادگیری برای لایه hidden 0.000001 تعریف شده است.

متغییر arch یک آرایه از سلول ها است که آن را درون تابع mlcnn قرار می دهیم.

به این ترتیب درون ساختار mlcnn به تابع mlcnn رفته و د آنجا بقیه توابع اجرا میشوند.

```
n = mlcnn(arch);
```

```
n.batchSize = 100;

n.costFun = 'xent';

n.nEpoch = 2;
```

batchSize تعداد پترن های ما است که قرار سات به صورت batch آنها را آموزش بدهیم.تعداد آن را ۱۰۰ در نظر گرفته ایم که باتوجه به ۴۰۰تا نمونه ۴ سری آموزش batch داریم.

Cost function را تابع xent در نظر گرفته ایم که برای عمل classification تابع مناسبی است.البته درون خود تابع cost انواع مختلف دیگر تابع وجوددارد که می توانیم از آن ها استفاده نماییم.

تعداد epoch هارا برابر ۲ در نظر گرفته ایم.

درون تابع mlcnn متغییر arch مقدار دهی اولیه می شود.البته توسط تابع init .سپس با دستور

```
arch = ensureArchitecture(self,arch)
```

صحت ساختار شبکه چک می شود.

همچنین در این قسمت چک میشود که آیا تمام پارامتر هایی که برای ساختار شبکه تعریف کردیم مقدار دهی اولیه شده اند یا نه و اگر مقداری نداشته باشند initialize میشوند.

تک تک لایه های input و conv وsubsamle و hidden و output مورد بررسی و مقدار دهی اولیه می شوند.در قسمت for درون تابع init ، arch تعریف شده را به صورت صحیح داخل شی ساخته شده از این شبکه mlcnn وارد میکند.

از آن جایی که آموزش ما به صورت batch است با self.layers{lL}=fmSize قراردادن،در واقع سایز ورودی را به شبکه میدهیم که ۲۰۰*۲۰۰ است.

تعداد فیچر مپ ها را با nFm بیان می کنیم. که مثلا برای لایه کانولوشن دوم برابر ۶ است.برای ورودی ۱ است و....

در ادامه همه پارامتر هایی که با init مقدار دهی شده اند را درون متغییر n قرار میدهیم.پارامترهای تعیین شده برای شبکه را مقدار دهی می کنیم.البته اگر خواستیم در demoMLCNN هم می توانیم پارمترهای دلخواه را بعدا تغییر دهیم.

در تابع زیر:

```
n = n.train(trainData,trainLabels);
```

شبکه را آموزش می دهیم.

داخل تابع train ، self را به عنوان پارامتر اول در نظر می گیرد و data و target را می گیرد و آموزش می دهد.

اول در هنگام شروع آموزش batchdata را تشکیل می دهیم.توسط تابع batchmake .

متغییر costbatch را هم برای محاصبه خطای هر batch تعریف می کنیم.

```
netInput = data(:,:,:,batchIdx);

netTargets = (targets(:,batchIdx));
```

netinput درواقع تمامی تعداد ورودی هایی است که به تعداد batch درون شبکه قرار می دهیم.nettarget هم برای خروجی است.سپس با تابع های زیر :

```
self = self.fProp(netInput, netTargets);

 self = self.bProp;

 self = self.updateParams;
```

محاسبات شبکه را انجام داده،خروجی را بدست آورده و خطا را بدست آورده و backprop می کنیم و وزن ها Update می کنیم.در قسمت fprop برای بدست آوردن مقدار کانولوشن از دستور convn (کانولوشن n بعدی) استفاده می کنیم.با تابع CalcAct مقدار خروجی را برای ما بسته به این که تابع فعال سازی مان tanh است و یا اصلا تابع فعال سازی اعمال نکرده ایم ،می دهد.از دستورstabilize برای اینکه مفداری ما محدود به یک رنج عددی مناسب باشند استفاده شده است.

هنگام فید فوروارد ما در لایه پولینگ downsampling اعمال خواهیم کرد به شبکه.که ما اینجا از میانگین گیری استفاده کرده ایم.برای این عمل درقسمت پولینگ اگر اندازه پنجره ما ۲*۲ است ابتدا آن پنجره را در ۰,۲۵ ضرب می کنیم و بعد از لایه کنولوشن کانولوشن گیری میکنیم با همین پنجره.درست مثل اینکه میانگین میگیریم از داده هایمان.

برای قسمت hidden و output هم محاسبات مانند mlp های معمولی خواهند بود. و مقدار خروجی را بدست خواهیم آورد.با استفاده از تابع CalcOutput مقدار خروجی را بدست می آوریم.سپس با توجه به نوع costfunction که تعریف کرده ایم مقدار خطای خروجی را بدست می آوریم.

در قسمت bprop ،خطا با دستور تابع bprop(self) به نرون های قبلی پروپگیت می شوند.برای لایه پولینگ آپدیت وزنی نخواهیم داشت و متغییر es خطای محلی بعدی ست که به قبلی می رسد.در قسمت up:

```
out = UP(self,data,scale);
```

عملیات downsampling را در واقع به نوعی برعکس انجام میدهیم و خطا را منتشر می کنیم.

در قسمت update patrameters وزن ها را با استفاده از دلتا هایی که در backprop حساب کرده بودیم آپدیت می کنیم و بعد از batch که تمام شد اموزش batch بعدی را شروع می کنیم.

بعد آموزش کل batch ها، توسط تابع self.assessNet از بین شبکه هایی که آموزش دادیم شبکه با کمترین خطا را ذخیره می کنیم.

با تابع traincost از همه ی خطاهای بدست آمده برای batch هایمان میانگین می گیریم. و این داده ها را به عنوان داده کنونی به تابع train cost می دهیم که یک ماتریس 1*2 است.چون ما فقط ۲ تا epoch تعریف کرده ایم.

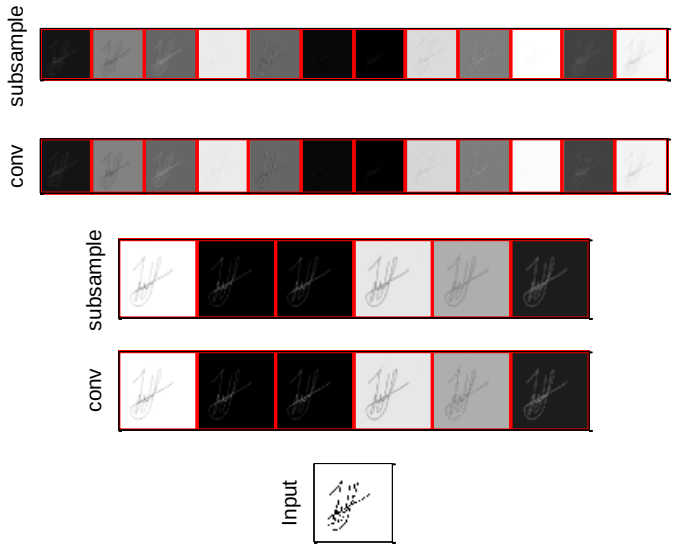
با دستور n.test ، testdata و testlable را به ورودی این تابع می دهیم و مقدار خروجی ClassErr است که به عنوان خطای داده های تست عنوان می شود.

داخل تابع test برخلاف train فقط قسمت fprop وجود دارد تا با تابع CalcActout مقدار خروجی را بدس آوریم.سپس با استفاده از تابع cost مقدار خطا را بدست می آوریم. Cost هزینه داده های تست به عنوان اولین خروجی برگردانده می شود.که در واقع همین برای ما مهم است.

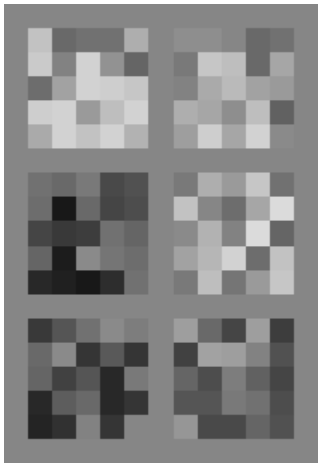
نتایج بدست آمده:

مقدار خطای بدست آمده برابر است با ۰,۵.

نتایج حاصل از visualization:



Conv. Layer 2



Conv. Layer 4