دانشکده مهندسی برق و رباتیک
مهندسی برق - الکترونیک

پایان‌نامه

برای دریافت درجه کارشناسی ارشد در رشته
الکترونیک، گرایش دیجیتال

عنوان

پیاده‌سازی ماشین بردار پشتیبان موازی (PSVM) بر روی FPGA

استاد راهنما
دکتر سید علی سلیمانی
استاد مشاور
دکتر علیرضا احمدی فرد
دانشجو
امین صاحبی

۱۳۹۲

نام خانوادگی: صاحبی نام: امین

عنوان: پیاده‌سازی ماشین بردار پشتیبان موازی (PSVM) بر روی FPGA

استاد راهنما: دکتر سید علی سلیمانی
استاد مشاور: دکتر علیرضا احمدی فرد

مقطع تحصیلی: کارشناسی ارشد رشته: الکترونیک گرایش: دیجیتال

دانشگاه: صنعتی شاهرود دانشکده مهندسی برق و رباتیک
تاریخ فارغ‌التحصیلی: ۱۳۹۲ تعداد صفحات: ۱۱۴

واژگان کلیدی: ماشین‌بردار پشتیبان، شناسایی چهره، $FPGA$، نرم مرتبه اول، کرنل

چکیده

در این پایان‌نامه، شبیه‌سازی و پیاده‌سازی مرحله‌ی آزمایش ماشین بردار پشتیبان موازی (PSVM) را ارائه می‌دهیم. مرحله‌ی آموزش به صورت جدا و از پیش توسط نرم افزار MATLAB صورت گرفته است و از نتایج آن در مرحله‌ی آزمایش استفاده گردیده است. در این پایان‌نامه از قطعه FPGA ساخت شرکت Xilinx به نام $Spartan3E$ استفاده شده که از طبقه‌ی FPGA های ارزان قیمت و دارای منابع محدود می‌باشد. به دلیل وجود منابع محدود در این قطعه از کرنل سخت‌افزار پسند استفاده شده است که با توجه به عدم نیاز آن به ضرب‌کننده‌ها باعث صرفه‌جویی در مصرف منابع می‌گردد. برای اجرای الگوریتم ماشین بردار پشتیبان موازی (PSVM) در معماری برخی بلوک‌های محاسباتی تغییراتی را ایجاد کرده‌ایم و نشان داده‌ایم که با پیاده‌سازی معماری پیشنهاد شده علی‌رغم کمی پیچیده‌تر شدن الگوریتم، تغییر محسوسی در عملکرد کلی سیستم از لحاظ سرعت انجام محاسبات، نسبت به سایر مراجع را شاهد خواهیم بود. ماشین بردار پشتیبان موازی با سرعت کلاک کمتر از $100\ MHz$ و با مصرف بخش بسیار کوچکی از منابع، بدون استفاده‌ی محسوس از ضرب‌کننده‌ها و تنها حدود 2% از منابع میکروپروسسور، نسبت به سایر مراجع ذکر شده بسیار سریعتر بوده و می‌تواند تعداد بسیار بیشتری فرایند را بصورت موازی در زمان بسیار کمتری انجام دهد.

تقدیم به

پدر و مادر عزیزم

خدایا...[1]

به من زیستنی عطا کن که در لحظه مرگ، بر بی‌ثمری لحظه‌ای که برای زیستن گذشته است، حسرت نخورم و مُردنی عطا کن که بر بیهودگیش، سوگوار نباشم. بگذار تا آن را، خود انتخاب کنم، اما آنچنان که تو دوست می‌داری.

تو می‌دانی و همه می‌دانند که شکنجه دیدن بخاطر تو، زندانی کشیدن بخاطر تو و رنج بردن به پای تو تنها لذت بزرگ زندگی من است، از شادی توست که من در دل می‌خندم، از امید رهایی توست که برق امید در چشمان خسته‌ام می‌درخشد و از خوشبختی توست که هوای پاک سعادت را در ریه‌هایم احساس می‌کنم. نمی‌توانم خوب حرف بزنم. نیروی شگفتی را که در زیر کلمات ساده و جمله‌های ضعیف و افتاده، پنهان کرده‌ام دریاب، دریاب.

تو می‌دانی و همه می‌دانند که زندگی از تحمیل لبخندی بر لبان من، از آوردن برق امیدی در نگاه من، از برانگیختن موج شعفی در دل من، عاجز است.

تو، چگونه زیستن را به من بیاموز، چگونه مردن را خود خواهم آموخت.

به من توفیق تلاش در شکست، صبر در نومیدی، رفتن بی‌همراه، جهاد بی‌سلاح، کار بی‌پاداش، فداکاری در سکوت، دین بی‌دنیا، مذهب بی‌عوام، عظمت بی‌نام، خدمت بی‌نان، ایمان بی‌ریا، خوبی بی‌نمود، گستاخی بی‌خامی، قناعت بی‌غرور، عشق بی‌هوس، تنهایی در انبوه جمعیت، و دوست داشتن بی‌آنکه دوست بداند، روزی کن.

اگر تنهاترین تنها شوم، باز خدا هست

او جانشین همه نداشتن‌هاست...

---

[1] مناجاتی از دکتر علی شریعتی.

سپاس گزاری...

سپاس خداوندگار حکیم را که با لطف بی‌کران خود، آدمی را زیور عقل آراست.

در آغاز وظیفه خود می‌دانم از زحمات بی‌دریغ استاد راهنمای خود، جناب آقای **دکتر سلیمانی**، صمیمانه تشکر و قدردانی کنم که قطعاً بدون راهنمایی‌های ارزنده ایشان، این مجموعه به انجام نمی‌رسید.

از جناب آقای **دکتر احمدی فرد**که زحمت مطالعه و مشاوره این رساله را تقبل فرمودند و در آماده سازی این رساله، به نحو احسن اینجانب را مورد راهنمایی قرار دادند، کمال امتنان را دارم.

در پایان، بوسه می‌زنم بر دستان خداوندگاران مهر و مهربانی، پدر و مادر عزیزم و بعد از خدا، ستایش می‌کنم وجود مقدس‌شان را و تشکر می‌کنم از برادران عزیزم به پاس عاطفه سرشار و گرمای امیدبخش وجودشان، که در این سردترین روزگاران، بهترین پشتیبان من بودند.

امین صاحبی
۱۳۹۲

فهرست مطالب

فهرست تصاویر

فهرست جداول

فصل ۱
مقدمه

۱.۱ پیشگفتار

از قرن نوزدهم به طور همزمان اما جداگانه از سویی نروفیزیولوزیست[۱]ها سعی کردند سامانه یادگیری و تجزیه و تحلیل مغز را کشف کنند و از سویی دیگر، ریاضیدانان تلاش کردند تا مدل ریاضی بسازند که قابلیت فراگیری و تجزیه و تحلیل عمومی مسائل را دارا باشند. اولین کوشش‌ها در شبیه‌سازی با استفاده از یک مدل منطقی توسط مک کلورک [۲] و والتر پیتز [۳] انجام شد که امروزه بلوک اصلی سازنده اکثر شبکه‌های عصبی مصنوعی است. این مدل فرضیه‌هایی در مورد عملکرد نرون‌ها ارائه می‌کند. عملکرد این مدل مبتنی بر جمع ورودی‌ها و ایجاد خروجی است. چنانچه حاصل جمع ورودی‌ها از مقدار آستانه بیشتر باشد اصطلاحاً نرون برانگیخته می‌شود. نتیجه این مدل اجرای توابع ساده مانند AND و OR بود. شبکه‌های عصبی مصنوعی دارای تاریخچه‌ای قدیمی و رنگارنگ است. در زمینه‌ی شبکه‌های عصبی مصنوعی مباحث متفاوت زیادی وجود دارد که هر کدام به صورت جداگانه و انفرادی طی سال‌ها توسط افراد مختلف ابداع، رشد و توسعه یافته‌اند که اکنون ما حاصل زحمات و تلاش آنها را ارج می نهیم.

اولین دیدگاه پیشرفته در زمینه‌ی شبکه‌های عصبی در دهه‌ی ۱۹۴۰ میلادی و با کار مک کلورک و والتر پیتز آغاز شد. آنها نشان دادند که شبکه و نرون‌های عصبی می توانند در اساس، هرگونه عملیات ریاضیاتی و منطقی را محاسبه کنند. کارهای آنها اغلب به عنوان پایه گذاران و بنیان گذاران شبکه‌های عصبی مصنوعی مورد تقدیر و احترام است.

---

[۱] Neuro Physiolozist
[۲] Mc Cholorch
[۳] Walter Pitz

۲.۱ تاریخچه

اولین برنامه‌ی عملی در رابطه با شبکه‌های عصبی مصنوعی در اواخر دهه‌ی ۱۹۵۰ میلادی با ابداع شبکه‌ی عصبی پرسپترون و قواعد آموزش آن توسط آقای فرانک روزنبلات[۱] معرفی شد. روزنبلات و همکارانش شبکه‌های عصبی مصنوعی پرسپترون را ابداع و آن را برای مقاصد شناسایی الگو مورد استفاده قرار دادند. این موفقیت تازه ایده‌ی اساسی و اصلی را در جهت تحقیقات گسترده و نوین در زمینه‌ی شبکه‌های عصبی مصنوعی برای محققان و دانشمندان ایجاد کرد.

متأسفانه مدتی بعد معلوم شدکه شبکه‌ی عصبی پرسپترون تنها می تواند مسائل بسیار ساده و محدودی را حل کند و دارای محدودیت‌های بسیاری است. در همان سال‌ها، دو محقق به نام‌های برنارد ویدرو [۲] و تد هاف[۳] یک روش آموزش جدیدی را معرفی کردند و از آن برای آموزش تطبیقی شبکه‌های عصبی خطی مورد استفاده قرار دادند که در ساختار و اساس بسیار شبیه به قواعد آموزش روزنبلات بود. آنها نام روش خود را آدلاین[۴] نام‌گذاری کردند. امروزه نیز قواعد آموزش ویدرو و هاف برای آموزش شبکه‌های عصبی مورد استفاده قرار می گیرد. اما متأسفانه شبکه‌های معرفی شده محدودیت‌های زیادی داشتند و نمی توانست از آنها برای مسائل پیچیده بهره جست[۱].

ویدرو و هاف هر دو از محدودیت‌های ذاتی شبکه‌های عصبی خود آگاه بودند و بنابراین به دنبال آن بودند تا شبکه‌ای ارائه دهند که بتوانند بر این محدودیت‌های ذاتی غلبه کند، گرچه سرانجام آنها نتوانستند به طور کاملاً موفقی الگوریتم‌های آموزشی خود را بهینه‌سازی و اصلاح کنند تا بتوانند شبکه‌های عصبی پیچیده‌تری را آموزش دهند.

اغلب محققین آن زمان عقیده داشتند که تحقیقات و جستجوی بیشتر در زمینه‌ی شبکه‌های عصبی به پایان خود رسیده است. این دیدگاه از آن ناشی می شد که هیچ ماشین محاسباتی قدرتمندی وجود نداشت که محققین بتوانند آزمایشات و تحقیقات خود را توسط آن پیاده‌سازی و آزمایش کنند. بنابراین به مدت یک دهه، تحقیقات و جستجوها در زمینه‌ی شبکه‌های عصبی مصنوعی کاملاً متوقف شد.

[۱] Frank Rosenblat
[۲] Bernard Widrow
[۳] Ted Haff
[۴] Adaline

در دهه‌ی ۱۹۷۰ میلادی، کارهای مهمی در زمینه‌ی شبکه‌های عصبی از سرگرفته شد. در سال ۱۹۷۲ کوهنن[۱] و اندرسن[۲] بصورت کاملاً مستقل و جداگانه، شبکه عصبی جدیدی را ارائه دادند که می توانست مانند حافظه عمل کند. همچنین در سال ۱۹۷۶ نیز استفن گروسبرگ[۳] شبکه‌ای تحت عنوان آوالانچ[۴] را برای تشخیص صحبت پیوسته و کنترل دست ربات مطرح کرد و در زمینه‌ی شبکه‌های خود سازمان یافته[۵] تحقیقات و نوآوری‌های زیادی را پایه ریزی کرد[۲].

همانطور که گفته شد، در دهه‌ی ۱۹۶۰ به دلیل وجود نداشتن ایده‌ی جدید برای رفع مشکلات شبکه‌های معرفی شده و همچنین به خاطر وجود نداشتن کامپیوتر محاسباتی قدرتمند که بتواند آزمایشات و محاسبات ریاضی را انجام دهد، علاقه‌مندی در زمینه‌ی شبکه‌های عصبی رو به افول گذاشت. اما در دهه‌ی ۱۹۸۰ میلادی این موانع و مشکلات مرتفع شد و به طور شگفت‌انگیزی تحقیقات و علاقه‌مندی‌ها در این زمینه اوج گرفت.

ماشین بردار پشتیبان[۶] یکی از روش‌های یادگیری بانظارت است که از آن برای طبقه‌بندی و رگرسیون استفاده می‌کنند. الگوریتم SVM اولیه در ۱۹۶۳ توسط ولادمیر وپنیک[۷] ابداع شدو در سال ۱۹۹۵ توسط وپنیک و کورینا کورتس[۸] برای حالت غیرخطی تعمیم داده شد. این روش از جمله روش‌های نسبتاً جدیدی است که در سال‌های اخیر کارایی خوبی نسبت به روش‌های قدیمی‌تر برای طبقه‌بندی از جمله شبکه‌های عصبی پرسپترون نشان داده است. مبنای کاری دسته‌بندی کننده SVM دسته‌بندی خطی داده‌ها است و در تقسیم خطی داده‌ها سعی می‌کنیم خطی را انتخاب کنیم که حاشیه اطمینان بیشتری داشته باشد. حل معادله پیدا کردن خط بهینه برای داده‌ها به وسیله روش‌های QP که روش‌های شناخته شده‌ای در حل مسائل محدودیت‌دار هستند صورت می‌گیرد. قبل از تقسیمِ خطی برای اینکه ماشین بتواند داده‌های با پیچیدگی بالا را دسته‌بندی کند داده‌ها را به وسیله تابعِ کرنل به فضای با ابعاد خیلی بالاترمی‌بریم. برای اینکه بتوانیم مساله ابعاد خیلی بالا را با استفاده از این روش‌ها حل کنیم از قضیه دوگانی لاگرانژ برای تبدیل مسالهٔ مینیمم‌سازی مورد نظر به فرم دوگانی آن که در آن به جای تابع پیچیده کرنل که ما را به فضایی با ابعاد بالا می‌برد، تابعِ ساده‌تری به نامِ تابع هسته که ضرب برداری تابع کرنل است ظاهر می‌شود استفاده می‌کنیم. از توابع هسته مختلفی از جمله هسته های نمایی(گاوسی)، چندجمله‌ای و سیگموید می‌توان استفاده نمود.

---

[۱] Kohenon
[۲] Anderson
[۳] Grosberg
[۴] Avalanch
[۵] Self Organization Map
[۶] Support Vector Machine
[۷] Vladimir Vapnik
[۸] Corinna Cortes

الگوریتم ماشین بردار پشتیبان، جز الگوریتم های تشخیص الگو دسته‌بندی می شود. از این الگوریتم در هر جایی که نیاز به تشخیص الگو یا دسته بندی اشیاء در کلاس‌های خاص باشد می توان استفاده کرد. در ادامه به کاربردهای این الگوریتم به صورت موردی اشاره می شود: سیستم آنالیز ریسک، کنترل هواپیما بدون خلبان، ردیابی انحراف هواپیما، شبیه سازی مسیر، سیستم راهنمایی اتوماتیک اتومبیل، سیستمهای بازرسی کیفیت، آنالیز کیفیت جوشکاری، پیش بینی کیفیت، آنالیز کیفیت کامپیوتر، آنالیز عملیاتهای آسیاب، آنالیز طراحی محصول شیمیایی، آنالیز نگهداری ماشین، پیشنهاد پروژه، مدیریت و برنامه‌ریزی، کنترل سیستم فرایند شیمیایی و دینامیکی، طراحی اعضای مصنوعی، بهینه سازی زمان پیوند اعضا، کاهش هزینه بیمارستان، بهبود کیفیت بیمارستان،مدیریت وجوه بیمه، دیریت سهام، تصویب چک بانکی، اکتشاف تقلب در کارت اعتباری، ثبت نسیه، بازبینی امضا از چکها، پیش بینی ارزش نسیه، مدیریت ریسک رهن، تشخیص حروف و اعداد، تشخیص بیماری و بسیاری دیگر کاربردها[۳].

در مورد تاریخچه آرایه سوئیچ فیوز های قابل برنامه‌ریزی چندباره، انقلابی نوین را در عرصه‌ی طراحی دیجیتال به وجودآورد که مفهوم طراحی دیجیتال را دچار تحولی عظیم در عرصه‌های دیدگاه معماری، حجم طراحی، سرعت و نوع نگرش به طراحی دیجیتال نموده است. طوری که امروزه FPGA ها (آرایه‌های گیتی قابل برنامه‌ریزی میدانی) یک بوم نقاشی سفید را در اختیار طراح قرار می‌دهد که به او اجازه می‌دهد تا طراحی دیجیتال خود را آن چنان که می‌خواهد و با هر حجم و پیچیدگی لازم، طراحی و سپس به جای انتخاب هایIC استاندارد و جدا از هم و کنار هم قرار دادن آنها بر روی یک مدار و وصل کردن آنها از طریق یک بورد مدار چاپی، با استفاده از یکی از زبان‌های توصیف سخت افزاری نظیر VHDL ، هر یک از قطعات دیجیتالی مورد نیاز را نوشته و با وصل کردن نرم‌افزاری آنها، سرانجام فایل تولید شده نهایی را از طریق یک رابط سخت افزاری بر روی یک بسته سخت افزاری خام با تعداد پایه‌های مورد نیاز برنامه‌ریزی کرده و از این IC جدید خود ساخته استفاده کند.

FPGA ها نسل جدید مدارهای مجتمع دیجیتال قابل برنامه‌ریزی هستند که عبارت FPGA از سرکلمه‌های **F**ield **P**rogrammable **G**ate **A**rray گرفته شده است. سرعت اجرای توابع منطقی در FPGA ها بسیار بالا و در حد نانوثانیه است. اگر بخواهیم FPGA ها را به طور ساده تشریح کنیم، عبارت است از یک تراشه که از تعداد بالایی بلوک‌های منطقی، خطوط ارتباطی و پایه‌های ورودی-خروجی تشکیل شده است که به صورت آرایه‌ای در کنار یکدیگر قرار دارند. خطوط ارتباطی که وظیفه آنها ارتباط بین بلوک‌های منطقی است از سوئیچ‌های قابل برنامه‌ریزی تشکیل شده‌اند. این سوئیچ‌ها بسته به نوعی که دارند، برخی تنها یک بار برنامه‌ریزی هستند و برخی به تعداد دفعات زیادی برنامه‌ریزی می‌شوند. بلوک‌های منطقی نیز دارای انواع مختلفی هستند که عموماً توسط المانی پایه، تمامی توابع منطقی را ایجاد می کنند. به عنوان مثال بلوک‌های منطقی در خانواده $ACT-1$ از شرکت Actel ، با پایه مالتی پلکسری عمل می‌کنند. به این معنا که توسط

مالتی پلکسر، توانایی ایجاد توابع منطقی مختلف را دارند. البته تعداد ورودی‌های هر بلوک منطقی متفاوت است و به نوع FPGA مربوط می‌شود. به عنوان مثال بلوک‌های منطقی در خانواده $ACT-1$ ، از نوع ۸ ورودی است. البته در برخی موارد به بلوک‌های منطقی سلول‌های منطقی نیز گفته می‌شود. البته بسیاری از سلول‌های منطقی بر اساس جداول LUT ساخته می‌شوند. LUT از تعدادی سلول‌های حافظه SRAM تشکیل می‌شود که در هنگام برنامه‌ریزی FPGA ، مقداردهی می‌شوند. به طور خلاصه LUT عبارتست از تولید توابع آماده برای استفاده در سلول‌های منطقی پیاده‌سازی توابع مختلف نیز به وسیله‌ی در کنارهم قرار گرفتن بلوک های منطقی و همچنین تنظیم ارتباط بین هر بلوک و به عهده گرفتن پردازش اطلاعات توسط هر بخش انجام می‌شود[۴].

۳.۱ انگیزه

ملاحظات سخت افزاری پیاده سازی شبکه های عصبی در ابعاد گسترده، وابسته به این است که چگونه بتوانیم بصورت مناسب و بهینه یک نرون تنها را پیاده سازی کنیم.
در سال های گذشته، حوزه‌ی تولید ادوات الکترونیکی شاهد پیدایش خانواده ای به نام FPGA ها بودند که انقلابی را در حوزه‌ی ادوات دیجیتالی بوجود آوردند. ویژگی مهم این ادوات بهینه بودن آنها می باشد و در حوزه‌ی کاربرد های دیجیتالی مدرن امروزه بهترین گزینه محسوب می شوند. پیاده سازی شبکه‌های عصبی مصنوعی بر روی FPGA ها با وجود تعداد بسیار زیاد نرون در این شبکه‌ها در حال حاضر مورد بحث و بررسی می باشد. در این نوشتار، بر روی موضوعاتی بحث می شود که شامل روش های پیاده‌سازی شبکه‌ی عصبی SVM به صورت خطی و غیرخطی به صورت ۲ کلاسه (باینری) و یا چند کلاسه می باشد. یکی از مهمترین چالش‌ها، برقراری یک موازنه بین سرعت و میزان منابع مصرفی موجود می باشد. به عنوان مثال استفاده از جداول جستجو، میزان سرعت را بهبود می بخشد اما میزان بیشتری از منابع را مصرف خواهد کرد. در پیاده‌سازی شبکه‌های عصبی که در آنها روابط غیر خطی حکم فرما می باشد استفاده از جداول جستجو، با وجود مصرف زیاد منابع، می‌تواند بهترین گزینه باشد. همچنین امکان پیاده‌سازی موازی شبکه‌های عصبی یکی از مهمترین مزایای این نوع ادوات دیجیتالی می باشد، زیرا شبکه‌های عصبی ذاتاً پتانسیل لازم برای اجرای محاسبات بصورت موازی را دارا می باشد.

معماری کامپیوتر امروزه به دلیل وجود ادوات متباین چند هسته ای به سرعت در حال تغییر می باشد. درست است که این ادوات چند هسته‌ای امکانات گسترده و مفیدی را در اختیار قرار می دهد و معماری کامپیوتر را توسعه داده است، اما نحوه‌ی استفاده از این منابع خود امروزه به یک مسئاله‌ی مهم و نسبتاً

پیچیده‌ای مبدل گشته است که باید با دقت حل گردد. FPGA ها ادواتی بسیار مستعد و کارآمد هستند که نحوه‌ی استفاده کردن کاربران با آنها نیز امری پیچیده نیست، همچنین این ادوات این قابلیت را دارند که کاربر بتواند واحدهای محاسباتی پیشرفته اضافی را برای انجام محاسبات خاص برای این ادوات تعریف کنند.

از طرفی، روش ماشین بردار پشتیبان یک روش مورد اعتماد و با کارآیی فوق العاده بالا در کلاسه‌بندی الگوها می باشد که در سال های اخیر به طور وسیعی در مسائل کلاسه‌بندی و رگرسیون خطی و غیرخطی نیز مورد استفاده قرار گرفته است و نتایج بسیار بهتری نسبت به روش های گوناگون موجود ارائه می دهد. از آنجا که ماشین بردار پشتیبان ذاتاً یک کلاسه‌بند باینری است، برای کلاسه بندی مسائلی با چند کلاس باید از چند کلاسه‌بند ماشین بردار پشتیبان و با کمک استراتژی‌های موجود استفاده کرد.

مسئله دسته‌بندی یکی از مهمترین مسائل مطرح شده در یادگیری ماشین است و بسیاری از مسائل را می توان به صورت یک مسئله دسته‌بندی کرده و حل کرد. از طرفی در یادگیری ماشین نیز روش‌های مختلفی برای حل مسئله دسته‌بندی صورت گرفته است.

یکی از روش‌هایی که در حال حاضر به صورت گسترده‌ای برای مسئله دسته‌بندی مورد استفاده قرار می گیرد، روش ماشین بردار پشتیبان (SVM) است. شاید به گونه‌ای بتوان محبوبیت کنونی روش ماشین بردار پشتیبان را با محبوبیت شبکه‌های عصبی در دهه گذشته مقایسه کرد. علت این قضیه نیز قابلیت استفاده این روش در حل مسائل گوناگون می‌باشد، در حالیکه روش‌هایی مانند درخت تصمیم‌گیری را نمی توان به راحتی در مسائل مختلف به کار برد.

برای ماشین بردار پشتیبان پیاده سازی‌های گوناگونی در طول سالیان اخیر تهیه شده است. اغلب این نرم افزارها در محیط‌های دانشگاهی طراحی و پیاده سازی شده‌اند، امادر این میان می‌توان محصولات تجاری را نیز پیدا کرد. نرم افزارهای گوناگون از لحاظ شرایط استفاده نیز با یکدیگر تفاوت دارند به گونه‌ای که برخی فقط برای استفاده‌های دانشگاهی و علمی قابل استفاده هستند و از برخی دیگر در کاربردهای تجاری نیز می‌توان استفاده کرد. با در نظر گرفتن تمامی موارد فوق و همچنین مواردی مانند سهولت استفاده و استفاده از روش‌های جدید در پیاده سازی نرم افزاری که در این پروژه انتخاب شده است، نرم افزار LIBSVM می‌باشد.

۴.۱ روش کلی پایان نامه

همانگونه که می دانید، بوردهای الکترونیکی دارای منابع محدودی می باشند لذا برای پیاده‌سازی الگوریتم‌های معرفی شده باید بسیار هوشیار و آگاه بود. بنابراین برای پیاده‌سازی ماشین بردار پشتیبان می بایست مرحله‌ی آموزش را به صورت جداگانه و به کمک نرم افزار انجام داد و سپس از نتایج آن استفاده کرده و برای اجرای مرحله‌ی آزمایش مورد استفاده قرار داد. بنابر این ما در این پایان‌نامه، مرحله‌ی آموزش کلاسه‌بند مورد نظر را به کمک نرم افزار MATLAB انجام داده و به کمک کد معتبر نوشته شده به نام LibSVM کلاسه بند خود را جهت یافتن بردارهای پشتیبان بکار برده و سپس نتایج بدست آمده را در الگوریتم پیشنهادی خود برای پیاده‌سازی بر روی FPGA بکار خواهیم برد. در این نوشتار سعی خواهیم کرد تا ساختار کلاسه‌بند را در مرحله‌ی آزمایش بصورت فرآیندی کاملاً موازی شده در آورده و بر روی سخت‌افزار مناسبی که جهت انجام فرآیندهای موازی بسیار مناسب است، پیاده‌سازی نماییم.

۵.۱ ساختار پایان نامه

پس از مقدمه، در فصل دوم این پایان نامه مروری بر پژوهش‌ها و تحقیقاتی مرتبط که در گذشته صورت پذیرفته‌اند آورده شده و عملکرد و نوآوری آنها مورد نقد و بررسی قرار خواهد گرفت.

در فصل سوم ،تئوری ماشین‌های بردار پشتیبان ارائه می شود. روش‌های بردار پشتیبان خطی و غیرخطی، به طور مفصلی شرح داده خواهد شد. حیله‌ی کرنل و همچنین توابع کرنل معرفی و بررسی خواهند شد و مزایا و معایب آنها نسبت به یکدیگر مورد بررسی قرار خواهند گرفت. سپس ماشین‌های بردار پشتیبان چند کلاسه مورد بررسی قرار می‌گیرد.

در فصل چهارم به معرفی و تشریح بورد FPGA خواهیم پرداخت و نحوه‌ی عملکرد آن در پردازش سیگنال دیجیتال را بیان خواهیم کرد. نحوه‌ی استفاده از قطعات داخلی FPGA ها و نیز برتری آنها نسبت به بوردهای تخصصی مشابه نظیر DSP ها آورده خواهند شد.

در فصل پنجم،از نتایج آموزش ماشین بردار پشتیبان در مرحله‌ی آموزش استفاده کرده و با الگوریتم معرفی شده مرحله‌ی آزمایش را ابتدا به کمک نرم افزار و سپس ساختار معرفی شده برای پیاده سازی آن را در محیط نرم افزار ISE مورد بررسی قرار خواهیم داد.

فصل ششم، نتایج شبیه سازی ساختار طراحی شده را نسبت به عملکرد حوزه‌ی نرم افزار مورد مقایسه قرار داده و طرز عملکرد الگوریتم ارائه شده را بر روی مجموعه داده‌های در دسترس را بررسی می کند. میزان بهبود عملکرد سیستم، مزیت روش ارائه شده بر روش‌های موجود، میزان منابع مصرفی در بورد FPGA از

مهمترین عواملی هستند که در این فصل ارائه می شوند.
در ادامه هم به ارائه‌ی نتایج و نتیجه گیری‌هایی که از روش ارائه شده برداشت شده است بیان خواهد شد، همچنین پیشنهادهایی برای ادامه‌ی این کار و پیشنهادهایی برای توسعه و پیشرفت الگوریتم بیان خواهد شد.
در انتهای این نوشتار بخش‌ها و موضوعاتی که به دلیل پیچیدگی و اجتناب از شلوغ شدن متن و دور شدن از بحث اصلی متن، در متن پایان نامه گنجانده نشده‌اند در ضمیمه‌هایی آورده شده‌اند باشد که مورد استفاده‌ی خواننده‌ی محترم قرار گیرد.

فصل ۲

تئوری ماشین‌های بردار پشتیبان

۱.۲ ماشین‌های بردار پشتیبان خطی

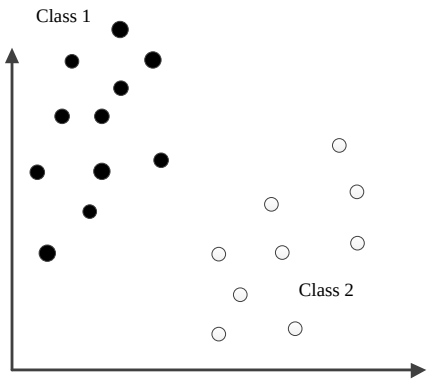
ماشین‌های بردار پشتیبان در سال ۱۹۹۵ توسط وپنیک و کرتس معرفی شدند. شبکه‌های عصبی SVM برخلاف سایر شبکه‌های عصبی که قبلاً مورد بحث و بررسی قرار گرفتند همانند شبکه‌های عصبی MLP و RBF و یا سایر شبکه‌های عصبی و یا شبکه‌های عصبی-فازی بجای محاسبه‌ی مقدار کمینه خطای مدل سازی و یا کلاسه‌بندی و یا طبقه‌بندی در مقابل، این شبکه‌ها ریسک عملیاتی را به عنوان تابع هدف در نظر گرفته و مقدار بهینه‌ی آن را محاسبه می‌کند.

یک مسئله‌ی کلاسه بندی خطی برای جداسازی داده های آموزشی دو کلاس را در نظر بگیرید. مجموعه‌ی $\{x_i, y_i\}$ را که شامل مجموعه بردارهای ویژگی مربوط به داده‌های آموزش باشند که به صورت خطی از یکدیگر تفکیک پذیر باشند را در نظر بگیرید. در این مجموعه داده‌های $x_i \in \Re^d$ عضو اعداد حقیقی d بعدی و مقادیر y نیز که برچسب های داده‌های آموزشی می باشند دارای مقادیر ۱ و ۱- می باشد. مقدار ۱ به معنای کلاس با برچسب ۱ و عدد ۱- به معنای کلاس با برچسب ۱- می باشد.

۱.۱.۲ بردارهای پشتیبان تفکیک پذیر خطی

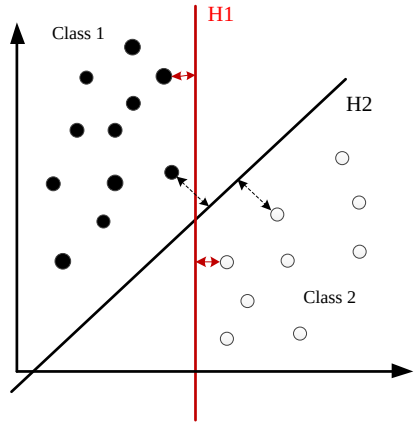
برای مثال ۲ کلاس زیر را که در شکل ۱.۲ نشان داده شده اند را در نظر بگیرید که در آن داده‌های سفید در کلاس ۱ و داده‌های سیاه در کلاس ۱- توزیع شده‌اند. اگر به کمک کلاسه‌بندی‌های متفاوت این ۲ کلاس را از هم جدا کنیم حالت های متفاوتی پیش می آید که این ۲ کلاس می‌توانند از هم بصورت کاملاً خطی [۱] از یکدیگر تفکیک شوند.

[۱] Hard Margin



شکل ۱.۲: نحوه‌ی توزیع کلاس‌های سفید و سیاه

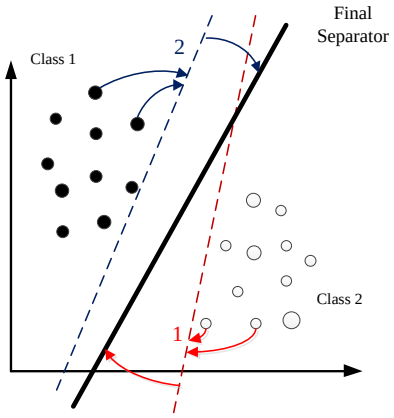
همانطور که در شکل آ.۲هم ملاحظه می‌کنید خطوط زیادی وجود خواهد داشت که این ۲ کلاس را می تواند از هم تفکیک کند. اما بحثی که در کلاسه‌بند بردارهای پشتیبان مطرح می‌شود این است که چگونه می‌توان بهترین خطی را یافت که بتواند بصورت خطی این ۲ کلاس را از هم تفکیک کند؟



شکل ۲.۲: جداسازی کلاس‌های سفید و سیاه به روش‌های مختلف

بهترین جواب را ماشین بردار پشتیبان به این سؤال می دهد، اگر بخواهیم بردارهای پشتیبان را به صورت ساده‌ای تشریح کنیم می‌توان گفت که این بردارهای پشتیبان در واقع همان نمونه‌های سفید یا سیاهی هستند که در ۲ کلاس درست در لبه‌ی مرز قرار گرفته‌اند و به طور کلی بردارهای پشتیبان نماینده‌ی مرزی هر کلاس هستند که با تشخیص دادن آنها می‌توان مرزهای هر کلاس را تشخیص داد و به عبارت با تشخیص مرزهای یک کلاس می توان به طور قطع کل یک کلاس را تشخیص داد همانگونه که یک کشور در حوزه‌ی جغرافیایی

به وسیله‌ی مرزهای آن کشور تشخیص داده می‌شود. با تشبیه بردار های پشتیبان به حوزه‌ی جغرافیایی می توان گفت بردارهای پشتیبان همانند مرزبانان یک کشور می‌باشند که با دفاع از مرزهای کشور خود، حدود کشور خود را نمایندگی می کنند و به هیچ خط جداکننده‌ای اجازه‌ی نزدیکی بیش از حد به مرزهای کشور خود را نمی دهند. بنابراین به زبان ریاضی نمونه‌هایی که در لبه‌ی مرز قرار گرفته‌اند اجازه‌ی نزدیکی خطوط جداکننده را در نزدیکی خود نخواهند داد و سعی می‌کنند که این خطوط در یک حاشیه‌ی مجازی از آنها قرار گیرند و به نوعی با وارد کردن نیرو آنها را از خود دور می‌کنند. در شکل ۳.۲ این تحمل‌ناپذیری بردارهای مرزی در حفظ و یا دفاع مرزبانان از کشور (کلاس) خود را ملاحظه می‌کنید.



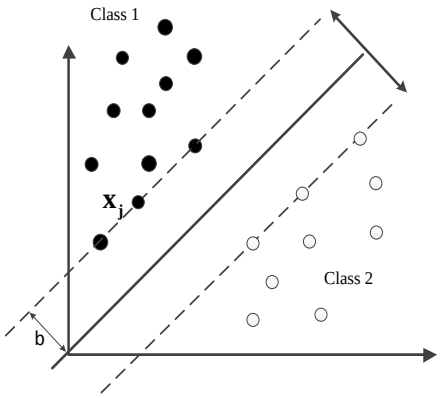
شکل ۳.۲: کش و قوس فراوان بین نمونه‌های مرزی هر کلاس

پس از کش و قوس فراوان بین نمونه‌های مرزی در حفظ حاشیه‌ی امنیت کافی از خط جداکننده به یک خط جداکننده‌ی یکتایی می‌رسیم که بهترین خطی است که ضمن جداکردن ۲ کلاس از یکدیگر نمونه‌های مرزی را نیز راضی نگه می‌دارد که در شکل ۴.۲ این حاشیه‌ی بهینه بدست آمده را ملاحظه می‌کنید.

اگر نیروی وارد شده از طرف نمونه‌های مرزی را وزن نمونه‌ها بنامیم معادله‌ی بهترین خط جداکننده را می‌توان به کمک این وزن ها را بدست آورد:

$$w_1x_1 + w_2x_2 + b = 0 \longrightarrow Line$$

$$w_1x_1 + w_2x_2 + w_3x_3 + b = 0 \longrightarrow Plane$$



شکل ۴.۲: بهترین خط جداکننده با بیشترین حاشیه‌ی ممکن

و در حالت کلی خواهیم داشت:

$$\sum_{i}^{\sharp sv} w_i x_i + b = 0 \tag{۱.۲}$$

و اگر به صورت برداری بنویسیم :

$$\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_3 \end{bmatrix} \quad , \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_3 \end{bmatrix} \tag{۲.۲}$$

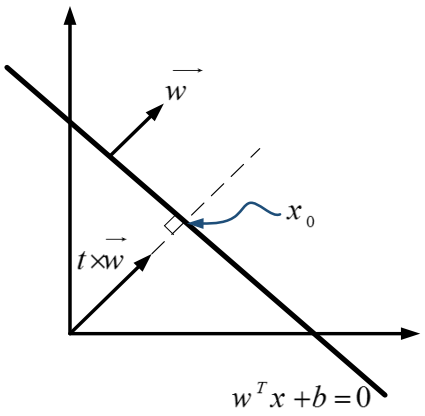
به طور کلی معادله‌ی تفکیک کننده‌ی خط جداساز به صورت زیر تعریف می شود:

$$\sum_{i}^{\sharp sv} w_i x_i + b = 0 \longrightarrow w^T x + b = 0 \tag{۳.۲}$$

باید خاطر نشان کرد به دلیل استفاده از حالت برداری است که ما از ترانهاده‌ی ماتریس وزن ها برای ضرب در نمونه‌ها استفاده می کنیم. همانطور که دیدیم حالت بهینه و ایده‌آل این است که این خط تفکیک کننده دارای بیشترین فاصله‌ی ممکن از نمونه‌های مرزی هر ۲کلاس باشد همانطور که در شکل ۴.۲ نشان داده شده است می‌خواهیم به کمک روابط ریاضی که در ادامه می‌آید این بیشترین حاشیه‌ی ممکن را که با b نشان داده شده است را بدست آوریم:

طبق شکل ۵.۲ اگر بخواهیم فاصله‌ی خط از مبدأ را بدست آوریم خواهیم نوشت:

$$\begin{cases} x_0 = t \times w \\ w^T x + b = 0 \longrightarrow t \times \mathbf{w^T} \times \mathbf{w} + b = 0 \end{cases} \tag{۴.۲}$$



شکل ۵.۲: نحوه‌ی بدست آوردن بیشترین حاشیه‌ی ممکن نسبت به مبدأ

از آنجا که حاصلضرب برداری وزن ها با مجذور مربعات آن برابر می باشد می توان $\mathbf{w^T \times w} = \| \mathbf{w^2} \|$ را در معادلات فوق جایگزین نمود. بنابراین در ادامه‌ی بدست آوردن بیشترین فاصله‌ی ممکن، می خواهیم بردار یکه‌ی t را محاسبه کنیم تا به کمک آن و با داشتن بردار مؤلفه‌ی بردار عمود بر بردار تصمیم $(w^T x+b=0)$ خواهیم توانست بیشترین اندازه‌ی ممکن برای b را بدست آوریم.

از نتیجه‌ی معادله‌ی ۴.۲ خواهیم داشت:

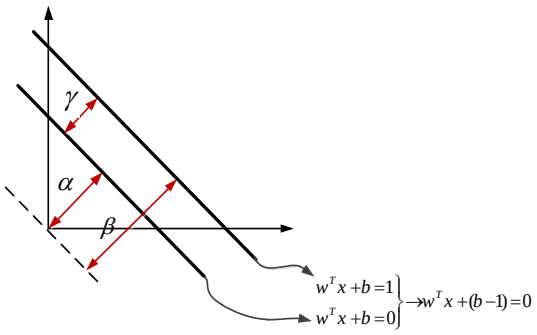
$$t = -\frac{b}{\| w \|^2} = -\frac{b}{w^T \times w} \tag{۵.۲}$$

با جایگذاری رابطه بدست آمده برای t می توان نوشت:

$$\begin{aligned} x_0 &= t \times w \\ &= -\frac{\mathbf{b}}{\| \mathbf{w} \|^{\mathbf{2}}} = -\frac{\mathbf{b}}{\mathbf{w^T \times w}} \times w \\ &= \frac{-b}{\| w \|} \end{aligned} \tag{۶.۲}$$

در نهایت طبق روابط بالا فاصله‌ی خط تصمیم از مبدأ $\| \frac{-b}{w} \|$ محاسبه شد.

اکنون می خواهیم طبق شکل ۴.۲ فاصله‌ی بین ۲ خط جداکننده را محاسبه کنیم، یعنی فاصله‌ی بین خطوط $w^T x + b = 1$ و $w^T x + b = 0$ را بدست آوریم قبلا در رابطه‌ی ۶.۲ فاصله‌ی آنها نسبت به مبدأ را محاسبه کردیم اگر ۲ خط را مطابق شکل زیر نسبت به هم در نظر بگیریم اگر فاصله‌ی خط $w^T x + b = 0$ را نسبت به مبدأ α و فاصله‌ی خط $w^T x + b = 1$ را نسبت به مبدأ β و فاصله‌ی بین ۲ خط را γ در نظر بگیریم می توان نوشت:



شکل ۲.۶: نحوه‌ی بدست آوردن بیشینه حاشیه‌ی ممکن

$$\gamma = \mid \beta - \alpha \mid \quad = \left| \frac{|b-1|}{\parallel w \parallel} - \frac{|b|}{\parallel w \parallel} \right| = \frac{1}{\parallel w \parallel} \tag{۷.۲}$$

بنابراین تابع هدفی که به دنبال آن بودیم به این گونه تعریف خواهد شد:

$$Max\, Distance = \frac{1}{\parallel w \parallel} + \frac{1}{\parallel w \parallel} = \frac{2}{\parallel w \parallel} \tag{۸.۲}$$

در واقع ما می خواهیم تا کسر $\frac{2}{\parallel w \parallel}$ بیشینه گردد و یا به عبارتی اندازه‌ی $\parallel w \parallel$ کمینه گردد. از آنجا که کمینه کردن یک کمیت خطی امکان پذیر نمی باشد، می توانیم به جای آن عبارت $\frac{1}{2}\parallel w \parallel^2$ و یا به عبارت بهتر $\frac{1}{2}w^T w$ را کمینه کنیم. در نهایت هدف نهایی ما کمینه کردن عبارت $\mathbf{\frac{1}{2}w^T w}$ این یعنی به تفکیک کننده‌ای برسیم که بیشترین حاشیه‌ی ممکن را داشته باشد، به شرطی که شرایط مسأله را هم ارضا کند. این شرایط به این ترتیب بیان می شوند:

$$\begin{cases} w^T x_i + b > 1 \\ w^T x_i + b < -1 \end{cases} \tag{۹.۲}$$

همانطور که از شرایط برداشت می شود جواب‌های مسئله باید یکی از ۲ کلاس را در بر بگیرد و نمی تواند متعلق به هیچ کلاسی نباشد. در نتیجه مسئله کلی بهینه سازی به صورت زیر خواهد شد:

$$\begin{aligned} &\min \quad \mathbf{\frac{1}{2}w^T w} \\ &Subject\ to: \\ &\begin{cases} \text{if} \quad y_i = 1 \rightarrow w^T x_i + b > 1 \ \forall i \\ \text{if} \quad y_i = -1 \rightarrow w^T x_i + b < -1 \ \forall i \end{cases} \end{aligned} \tag{۱۰.۲}$$

در ابتدای کار برای حل مسئله باید شرایط مسئله را کمی ساده تر و گویاتر کنیم، شرایطی که در رابطه‌ی ۱۰.۲ ذکر شدند بسیارکلی و ناگویا می باشند لذا با کمی تغییرات شرایط جدید را خواهیم داشت:
از آنجا که در تعریف مسئله ذکر کردیم که $y_i \in \{+1, -1\}$ لذا با ضرب ۱+ در شرط اول تغییری حاصل نمی‌شود ولی با ضرب ۱- در شرط دومی، شرایط کاملاً شبیه شرط اول خواهد شد. بنابراین ۲ شرط ما به یک شرط کاهش می یابد و کمی شرایط صورت مسئله گویاتر می گردد.

$$\xrightarrow{\text{ضرب طرفین در } y_i} \left\{ \begin{array}{l} y_i \times (w^T x_i + b > 1) \\ y_i \times (w^T x_i + b < -1) \end{array} \right\} \rightarrow y_i(w^T x + b) - 1 \geqslant 0 \qquad (۱۱.۲)$$

در نهایت معادله‌ی تفکیک کننده‌ی مورد نظر ما به این صورت می گردد:

$$\begin{array}{l} \min \quad \mathbf{\frac{1}{2} w^T w} \\ \\ \textit{Subject to}: \\ y_i(w^T x_i + b) - 1 \geqslant 0) \; \forall i \end{array} \qquad (۱۲.۲)$$

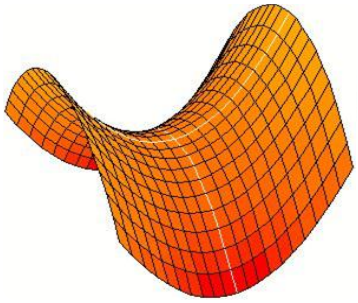
برای حل این مسئله به این گونه که هست قطعاً به بن بست خواهیم خورد زیرا تعداد مجهول های مسئله بیشتر از معلومات مسئله هستند از جمله مجهول بایاس (b) که تنها شرایط حل ریاضی مسئله را با بن بست مواجه می کند وگرنه حذف آن به هر صورت ممکن به کیفیت مطلوب نهایی ما خللی وارد نخواهد کرد.
برای حل چنین مسائل ریاضی و برای خارج کردن مسئله‌ی تعریف شده از بن بست، از معادلات لاگرانژ بهره خواهیم برد. به این ترتیب که برای شرط تعریف شده یک ضریب نامنفی α در نظر می گیریم و تابع هدف را اینگونه تغییر می دهیم.

$$\boldsymbol{\ell_p} = \frac{1}{2} w^T w - \underbrace{\sum_i \alpha_i \left[y_i(w^T x_i + b) - 1 \right]}_{\text{یک عدد مثبت}} \qquad (۱۳.۲)$$

کم شدن یک عدد مثبت از $\mathbf{\frac{1}{2} w^T w}$ به ما در جهت رسیدن به هدف اصلی و حل تابع هدف کمک شایانی خواهد کرد. هدف ما به اینصورت تغییر کرد که اکنون ما می خواهیم $\boldsymbol{\ell_p}$ کمینه گردد. در تابع هدف اگر قسمتی که مشخص شده است یک عدد مثبت باشد متناسب با مقادیر پارامترهای مسئله عدد مثبتی تولید می گردد که مقدار $\boldsymbol{\ell_p}$ را کمینه می کند، به عبارت دیگر می توان گفت $\alpha_i \left[y_i(w^T x_i + b) - 1 \right]$ نقش تابع جریمه را خواهد داشت.

$$\boldsymbol{\ell_p}$$ نسبت به α باید max گردد $\Rightarrow \Leftarrow$ نسبت به w و b باید min شود

تابعی که به ازای یک سری دسته متغییرها max و به ازای دسته دیگری min شود اصطلاحاً نقطه زینی[۱] گفته می‌شود.
در شکل زیر نمونه‌ای از یک نقطه زینی آورده شده است.



شکل ۷.۲: نمونه‌ای از یک نقطه‌ی زینی

با توجه به رابطه‌ی ۱۳.۲ اینگونه به نظر می‌رسد که وضعیت حل مسئله سخت تر شده است. تفاوت صورت مسئله جدید نسبت به قبلی در این است که در رابطه‌ی ۱۳.۲ نسبت به رابطه‌ی ۱۰.۲ به جای پرداختن به w وb به حل α_i و بعد از یافتن α_i ها به سراغ w و b رفته و آنها را راحت تر پیدا می کنیم.
برای یافتن α_i ها باید به نحوی w و b را از رابطه‌ی ۱۳.۲ حذف کنیم. برای این منظور داریم:

$$\frac{\partial \ell_p}{\partial w} = 0 \qquad , \frac{\partial \ell_p}{\partial b} = 0 \tag{۱۴.۲}$$

اگر از معادله‌ی اصلی نسبت به w و b و مشتقات جزئی بگیریم می توان به کمک آنها w و b را حذف کرد که با اجرای این روش قید دیگری به شرایط مسئله اضافه خواهد شد:

$$\frac{\partial \ell_p}{\partial w} = w - \sum_i \alpha_i y_i x_i = 0 \longrightarrow \mathbf{w} = \sum_{\mathbf{i}} \alpha_{\mathbf{i}} \mathbf{y_i} \mathbf{x_i} \tag{۱۵.۲}$$

$$\frac{\partial \ell_p}{\partial b} = 0 - \sum_i \alpha_i y_i = 0 \longrightarrow \sum_{\mathbf{i}} \alpha_{\mathbf{i}} \mathbf{y_i} \tag{۱۶.۲}$$

[۱] Saddle Point

با اعمال مشتقات جزئی می توان مقدار w را نیز بدست آورد که در ۱۶.۲ نشان داده شده است.
با جایگذاری این مشتقات در $\boldsymbol{\ell_p}$، مسئله‌ي ما به مسئله‌ی دومی تبدیل می گردد که به کمک ضرایب لاگرانژ قابل حل می باشد:

$$\begin{aligned}\boldsymbol{\ell_{\mathbf{Dual}}} &= \frac{1}{2}w^T w - \sum_i \alpha_i[y_i(w^T x_i + b) - 1] \\ &= \frac{1}{2}(\sum_i(\alpha_i y_i x_i))^T(\sum_i \alpha_i y_i x_i) - \sum_i \alpha_i[y_i(\sum_j \alpha_j y_j x_j^T x_j + b) - 1]\end{aligned} \tag{۱۷.۲}$$

که با کمی ساده سازی و کمی عملیات جبری به مسئله‌ی زیر خواهیم رسید:

$$\boldsymbol{\ell_{\mathbf{Dual}}} = \frac{-1}{2}\sum_i\sum_j \alpha_i\alpha_j y_i y_j x_i^T x_j + \sum_i \alpha_i \tag{۱۸.۲}$$

در نهایت مسئله‌ای که داشتیم به مسئله‌ی جدیدی برحسب ضرایب لاگرانژ تبدیل گردید، حال برای رسیدن به هدف خود بایستی این مسئله را نسبت به α_i بیشینه کنیم.
صورت مسئله‌ی نهایی در زیر آمده است:

$$\begin{aligned}&\max \quad \tfrac{-1}{2}\textstyle\sum_i\sum_j \alpha_i\alpha_j y_i y_j x_i^T x_j + \sum_i \alpha_i \\ &\textit{Subject to}: \\ &\textstyle\sum_i \alpha_i y_i = 0 \quad , \alpha_i \geqslant 0\end{aligned} \tag{۱۹.۲}$$

می توانیم در عبارت فوق به دلیل وجود ضریب منفی $\frac{-1}{2}$ کل عبارت را در منفی ضرب کرده سپس بجای max ، min را محاسبه کنیم.
اگر بخواهیم صورت مسئله را بصورت ماتریسی در آوریم خواهیم داشت:

$$\frac{1}{2}\sum_i\sum_j h_{ij}\alpha_i\alpha_j = \boldsymbol{\alpha^T \mathbf{H} \alpha} \tag{۲۰.۲}$$

$$\boldsymbol{\alpha} = \begin{bmatrix}\alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_3\end{bmatrix} \quad \mathbf{H} = \begin{bmatrix}h_{11} & h_{12} & \dots & h_{1n} \\ h_{21} & h_{22} & \dots & h_{2n} \\ \dots & \ddots & \dots & \vdots \\ h_{n1} & h_{n2} & \dots & h_{nn}\end{bmatrix} \tag{۲۱.۲}$$

در نهایت به رابطه‌ی درجه دومی رسیدیم که اکنون می توانیم آن را حل کنیم. با نوشتن کامل صورت مسئله به شکل ماتریسی به همراه قیود مسئله رابطه‌ی زیر را داریم:

$$\begin{aligned} &\min \quad \tfrac{1}{2}\alpha^T H\alpha + f^T\alpha \\ &\textit{Subject to}: \\ &\textstyle\sum_i \alpha_i y_i = 0 \quad , \alpha_i \geqslant 0 \end{aligned} \tag{۲۲.۲}$$

که در آن ماتریس H به صورت یک ماتریس $n \times n$ و ماتریس f بصورت یک ماتریس $n \times 1$ می باشد که همه‌ی درایه‌های آن ۱- می باشند.

برای محاسبه پارامترهای مسئله بایستی w و b را محاسبه کنیم. w که همان بردار وزن‌ها می باشد را از روابط قبل محاسبه کردیم، که برابر بود با $w = \sum_i \alpha_i y_i x_i$ ، اما برای محاسبه پارامتر b بایستی به سراغ شرایط KKT که در رابطه‌ی ۱۲.۲ به آن اشاره شد، برویم:

طبق شرایط مسئله داشتیم $\sum_i \alpha_i y_i \geqslant 0$ ،از همین رابطه برای محاسبه b استفاده می نماییم:

$$\alpha_i[y_i(w^T x_i + b) - 1] = 0 \tag{۲۳.۲}$$

در این رابطه حاصلضرب ۲ عبارت مثبت را داریم که حاصل آن برابر با صفر شده است لذا یا بایستی α برابر با صفر باشد و یا حاصل $y_i(w^T x_i + b) - 1$ برابر باصفر گردد. لذا در ادامه این ۲ حالت را بررسی خواهیم کرد.

از آنجا که در قیود رابطه‌ی ۲۲.۲ داشتیم $\alpha_i \geqslant 0$ لذا می توان نتیجه گرفت تنها می توان مقادیر α های مثبت را در نظر گرفت، پس می توان نوشت:

$$\begin{aligned} y_i(w^T x_i + b) - 1 &= 0 \\ y_i(w^T x_i + b) &= 1 \end{aligned} \tag{۲۴.۲}$$

همانطور که از صورت مسئله هم می دانیم مقادیر برچسب‌های هر کلاس یا ۱+ یا ۱- می باشد، بنابراین با ضرب طرفین معادله‌ی ۲۴.۲ خواهیم داشت:

$$y_i^2(w^T x_i + b) = y_i \tag{۲۵.۲}$$

پر واضح است که مجذور عددهای ۱- و ۱+ به عنوان برچسب‌های کلاس ها، ۱+ خواهد شد بنابراین رابطه ساده‌تر خواهد شد:

$$w^T x_i + b = y_i \tag{۲۶.۲}$$

طبق تعریف می دانیم بردارهای پشتیبان آن دسته از نمونه‌هایی از هر کلاس می باشند به ازای هرکدام از آنها شرایط مسئله ارضا گردد، به عبارت بهتر اگر **S** را مجموعه بردارهای پشتیبان بدانیم خواهیم داشت:

$$\mathbf{S} = \{\, i \,|\, \alpha_i > 0 \,\} \tag{۲۷.۲}$$

بنابراین طیق مجموعه بردارهای پشتیبان که در بالا به آن اشاره شد، آن دسته از α ها که در مجموعه‌ی **S** باشند، وقتی در رابطه‌ی ۲۶.۲ قرار گیرند یا بر روی خط ۱- قرار می گیرند و یا بر روی خط ۱+! از همین برای محاسبه‌ی مقدار b استفاده خواهیم کرد.
از رابطه‌ی ۲۶.۲ استفاده خواهیم نمود:

$$\begin{aligned} i \in \mathbf{S} \Longrightarrow \quad b &= y_i - w^T x_i \\ &= \frac{1}{|\mathbf{S}|} \sum_{i \in \mathbf{S}} b_i \\ &= \frac{1}{|\mathbf{S}|} \sum_{i \in \mathbf{S}} (y_i - w^T x_i) \end{aligned} \tag{۲۸.۲}$$

بدین ترتیب مقادیر پارامترهای مورد نیاز کلاسه‌بند یعنی مقادیر w و b را نیز محاسبه کردیم. اگر بخواهیم کل مسئله را بصورت خلاصه بیان کنیم می توان روابط زیر را نوشت:

$$\left.\begin{array}{l} \min \quad \frac{1}{2}\alpha^T H \alpha + f^T \alpha \\ \\ \textit{Subject to}: \\ \sum_i \alpha_i y_i = 0 \quad , \alpha_i \geqslant 0 \end{array}\right\} \longrightarrow w = \sum_{i \in \mathbf{S}} \alpha_i y_i x_i \quad , b = \frac{1}{|\mathbf{S}|} \sum_{i \in \mathbf{S}} (y_i - w^T x_i) \tag{۲۹.۲}$$

اکنون مقادیر w و b رو داریم رابطه‌ی آزمایش کلاسه بند خود را بیان می کنیم تا ببینیم آیا می توان به کمک مقادیر بدست آمده، تابع آزمایش بردار پشتیبان را معادل سازی کرد یا خیر؟ از آنجا که می دانیم برچسب های هر کلاس ۱+ و یا ۱- می باشد و به کمک رابطه‌ی ۱۲.۲ خواهیم داشت:

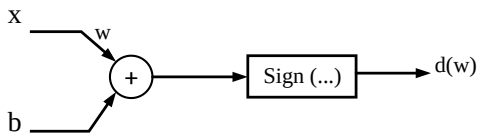
$$y_i = +1 \longrightarrow w^T x_i + b > +1 > 0$$

$$y_i = -1 \longrightarrow w^T x_i + b < -1 < 0$$

ملاحظه می کنیم که تابع آزمایش بردار پشتیبان می تواند مشابه یک تابع sign عمل کند بطوریکه داشته باشیم:

$$y_i = sign(w^T x_i + b) \tag{۳۰.۲}$$

بنابراین می توان این تابع آزمایش را همانند یک نرون شبکه‌ی عصبی عمل کرد همانگونه که در شکل زیر هم ملاحظه می شود با ضرب ورودی ها در بردار وزن w و به همراه b که همان بایاس شبکه عصبی می باشد و سپس عبور آنها از تابع علامت، یک نرون کامل شبکه عصبی را می سازد همانطور که در شکل زیر هم دیده می شود:



شکل ۸.۲: تابع آزمایش بردار پشتیبان به عنوان یک نرون تنها

مسائلی که تا کنون بحث کردیم در صورتی محقق می شد که امکان تفکیک پذیری کاملاً خطی بین نمونه ها وجود داشت، به بیان دیگر کلاسه‌بندی که تا کنون داشتیم اجازه‌ی هیچ گونه تخطی را به اعضای گروه نمی داد، در غیر اینصورت کلاسه بندی که معرفی کردیم با روبطی که ذکر شد کارآیی نخواهد داشت زیرا نمی‌توان α_i هایی پیدا کرد که محدود بوده و بتوانند شرایط مسئله را ارضا کنند. در مسائلی که امکان تفکیک پذیری کاملاً خطی وجود ندارد از بردارهای پشتیبان با امکان تفکیک پذیری غیر خطی استفاده خواهیم نمود.

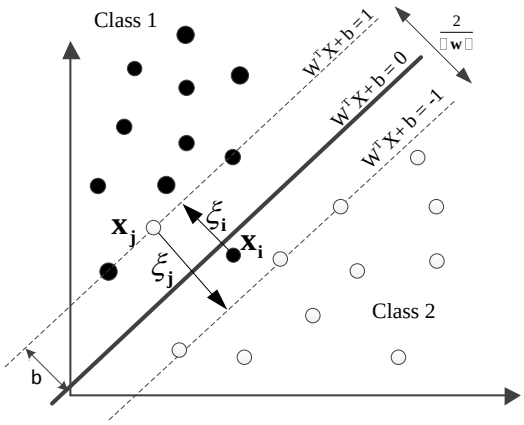
۲.۱.۲ بردارهای پشتیبان تفکیک پذیر غیرخطی

همانگونه که در شکل زیر ملاحظه می شود از هر کلاس یک عضو تخطی کرده و وارد حوزه‌ی دیگری شده است، که در شکل آنها را با x_i و x_j نشان داده شده است. حال می‌خواهیم این حالت را بررسی کرده و معادلات ۱۰.۲ را برای این حالت بررسی کنیم.

$$\begin{cases} \text{if} \quad \mathrm{y}_i = 1 \rightarrow \mathrm{w}^T x_i + b > 1 \; \forall i \\ \text{if} \quad \mathrm{y}_i = -1 \rightarrow w^T x_i + b < -1 \; \forall i \end{cases} \tag{۳۱.۲}$$

ملاحظه خواهیم کرد که بدلیل ۲ عنصری که به عنوان نمونه از مرز خود تخطی کرده‌اند، رابطه‌ی ۳۱.۲ برای آن نمونه ها برقرار نمی‌باشد لذا برای آن دسته از نمونه‌هایی که مرتکب خطا شده‌اند باید یک ضریب جریمه‌ای در نظر گرفت و آن را به معادلات بالا اعمال کرد. در نتیجه رابطه‌ی فوق را بازنویسی کرده و می‌نویسیم:

$$\begin{cases} \text{if} \quad \mathrm{y}_i = 1 \rightarrow w^T x_i + b > 1 - \zeta_\mathbf{i} \quad \forall i \\ \text{if} \quad \mathrm{y}_i = -1 \rightarrow w^T x_i + b < -1 + \zeta_\mathbf{i} \quad \forall i \end{cases} \tag{۳۲.۲}$$



شکل ۹.۲: تخطی یک عضو از هر کلاس و وارد شدن به کلاس دیگر

اگر بخواهیم روابط را بطور خلاصه‌تری دوباره بنویسیم می توان ۲ طرف رابطه را در y_i ضرب نمود در اینصورت خواهیم داشت:

$$y_i(w^T x_i + b) \geq 1 - \zeta_i \tag{۳۳.۲}$$

بنابراین به دلیل اینکه شرایط مسئله تغییر کرده است، باید به نوعی عنصری را که خارج از حوزه‌ی مورد نظر قرار گرفته است باید جریمه شود ضریب جریمه را C نشان داده و آن را در مسئله وارد می نماییم، لذا مسئله‌ی ما به اینصورت تغییر خواهد کرد:

$$\begin{aligned} &\min \quad \tfrac{1}{2} w^T w + C \textstyle\sum_i \zeta_i \\ &\textit{Subject to}: \\ &y_i(w^T x_i + b) \geq 1 - \zeta_i \quad \forall i \\ &\zeta_i \geq 0 \end{aligned} \tag{۳۴.۲}$$

حال باید همانند روال گذشته مسئله را توسط ضرایب لاگرانژ به مسئله‌ی دیگری تبدیل نماییم که قابل حل باشد. در اینجا کمی روابط پیچیده‌تر می شوند زیرا تفکیک پذیری غیرخطی است و ۲ ضریب لاگرانژ برای حل چنین مسئله‌ای مورد نیاز است، می توان نوشت:

$$\begin{aligned} \ell_{\mathbf{P}} = \frac{1}{2} w^T w \quad &+ \quad C \sum_i \zeta_i \\ &- \quad \sum_i \alpha_i [y_i(w^T x_i + b) - 1 + \zeta_i] \\ &- \quad \sum_i \mu_i \zeta_i \end{aligned} \tag{۳۵.۲}$$

با زهم یک نقطه‌ی زینی داریم که نسبت به یک سری از داده‌ها بیشینه و نسبت به یک سری دیگر از داده‌ها کمینه می‌باشد.

نسبت به α_i و μ_i باید بیشینه گردد $\Rightarrow \boldsymbol{\ell_p} \Leftarrow$ نسبت به w و b و ζ_i باید کمینه گردد

همانطور که در بخش قبلی معدله‌ی لاگرانژ مربوط را حل کردیم بار دیگر سراغ مشتقات جزئی رفته و مشتقات را برحسب مجهولات مسئله بدست آورده و برابر صفر قرار می دهیم تا بتوانیم α_i و μ_i را بدست آوریم.

$$\frac{\partial \ell_p}{\partial w} = 0 \longrightarrow w = \sum_i \alpha_i y_i x_i \tag{۳۶.۲}$$

$$\frac{\partial \ell_p}{\partial b} = 0 \longrightarrow \sum_i \alpha_i y_i = 0 \tag{۳۷.۲}$$

$$\frac{\partial \ell_p}{\partial \zeta_i} = 0 \longrightarrow C - \alpha_i - \mu_i = 0 \longrightarrow \alpha_i + \mu_i = C \tag{۳۸.۲}$$

می دانیم که در معادلات لاگرانژ می‌بایستی پارامترهای α_i و μ_i بزرگتر از صفر باشند، و از آنجا که در رابطه‌ی ۳۷.۲ شرط تازه‌ای به شروط مسئله اضافه شده است می‌توان نتیجه گرفت:

$$0 \leq \alpha_i \leq C$$
$$0 \leq \mu_i \leq C \tag{۳۹.۲}$$

همانطور که ملاحظه می شود در بخش قبلی ضرایب α_i ها محدود نبودند ولی در این بخش بدلیل وجود ضریب جریمه برای نمونه‌های خطاکار این ضرایب محدود می باشند.
بنابر نتایج گرفته شده صورت کلی مسئله ما چنین خواهد شد:

$$\begin{aligned} &\min \quad \tfrac{1}{2}\textstyle\sum_i \sum_j \alpha_i \alpha_j y_i y_j x_i^T x_j - \sum_i \alpha_i \\ &\textit{Subject to}: \\ &\textstyle\sum_i \alpha_i y_i = 0 \\ &0 \leq \alpha_i \leq C \quad \forall i \end{aligned} \tag{۴۰.۲}$$

می‌توان اینگونه به مسئله نگاه کرد که ضریب C در بخش قبلی که تفکیک پذیری خطی بود بینهایت بود $C = \infty$، در حالیکه در مسئله‌ی کنونی که تفکیک پذیری غیر خطی می‌باشد ضریب جریمه‌ی C محدود می

باشد.

همانند قبل، برای محاسبه پارامترهای مسئله بایستی w و b را محاسبه کنیم. w که همان بردار وزن‌ها می باشد را از روابط قبل محاسبه کردیم، که برابر بود با $w = \sum_i \alpha_i y_i x_i$، اما برای محاسبه پارامتر b بایستی به سراغ شرایط KKT که در رابطه‌ی ۱۲.۲ به آن اشاره شد، برویم اما با این تفاوت که شرایط مسئله در این جا تابعی از ضریب جریمه‌ی ζ_i می باشد کمی شرایط متفاوت می‌باشد:

$$\alpha_i[y_i(w^T x_i + b) - 1 + \zeta_i] = 0 \quad (۴۱.۲)$$

$$\mu_i \zeta_i = (C - \alpha_i)\zeta_i = 0 \quad (۴۲.۲)$$

با توجه به روابط فوق ۲ حالت برای α_i وجود خواهد داشت، یا α_i برابر صفر خواهد بود و یا باید برابر با C باشد. در هر ۲ حالت موارد را بررسی خواهیم کرد:

اگر $\alpha_i = 0$ خواهیم داشت:

$$\mu_i = C \longrightarrow \zeta_i = 0 \quad (۴۳.۲)$$

$$y_i(w^T x_i + b) - 1 \geq 0 \quad (۴۴.۲)$$

از این روابط آن دسته از نمونه‌هایی که جزء بردارهای پشتیبان نمی باشند بدست مشخص خواهند شد. حال اگر شرط دوم را نیز بررسی کنیم یعنی حالتی را که $\alpha_i = C$ خواهیم داشت:

$$\mu_i = 0 \longrightarrow \zeta_i \geq 0 \quad (۴۵.۲)$$

$$y_i(w^T x_i + b) - 1 + \zeta_i = 0 \quad (۴۶.۲)$$

که در نتیجه‌ی این رابطه آن دسته از نمونه‌هایی مشخص خواهند شد که تخطی کرده‌اند. حالت دیگری که وجود دارد، زمانی است که ضریب α_i نه برابر با صفر باشد و به برابر با C یعنی داشته باشیم $0 \leq \alpha_i \leq C$، این حالتی است که بردارهای پشتیبان مشخص می شوند:

$$0 < \alpha_i < C \longrightarrow 0 < \mu_i < C \longrightarrow \zeta_i = 0 \quad (۴۷.۲)$$

$$\Longrightarrow y_i(w^T x_i + b) - 1 = 0 \longrightarrow y_i(w^T x_i + b) = 1 \quad (۴۸.۲)$$

در صورتی که دو طرف رابطه‌ی ۴۸.۲ را در y_i ضرب نماییم رابطه‌ی زیر بدست خواهد آمد که رابطه‌ی نهایی بردارهای پشتیبان می‌باشد.

$$w^T x_i + b = y_i \quad (۴۹.۲)$$

حال که بردارهای پشتیبان مشخص شده‌اند، برای کامل شدن حل روابط تنها پارامتر b را محاسبه نکرده‌ایم که آن را هم می توانیم به کمک رابطه‌ی ۵۰.۲ بدست آورد:

$$b = y_i - w^T x_i \qquad (۵۰.۲)$$

در پایان بایستی ذکر شود که پارامتر C که عامل جریمه کننده‌ی نمونه‌های خطاکار می باشد به طور تجربی تنظیم می گردد و روش و قاعده‌ی مشخصی در این زمینه برای پیدا کردن بهترین مقدار برای ضریب C وجود ندارد. البته می توان به کمک برخی الگوریتم های بهینه سازی مانند الگوریتم PSO و یا الگوریتم بهینه سازی ژنتیک و یا الگوریتم های استعماری مقدار بهینه‌ی آن را بدست آوریم.

۲.۲ ماشین‌های بردار پشتیبان غیر خطی

بطور اساسی، ایده‌ی بردارهای پشتیبان حول محور ۲ رابطه‌ی ریاضیاتی می چرخد که در زیر معرفی شده‌اند:

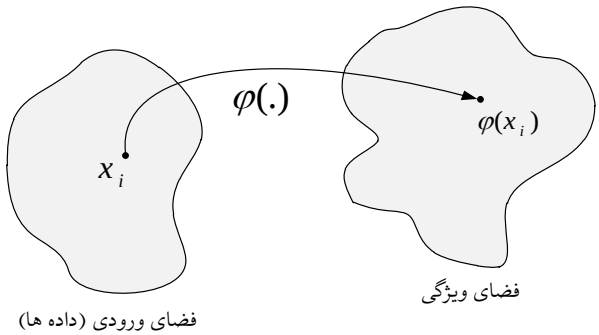
۱. نگاشت غیر خطی بردارهای ورودی به یک فضای ویژگی با ابعاد گسترده که هم از نظر ورودی‌ها و هم از دید خروجی پنهان می‌باشد.

۲. ساختن یک ابر صفحه‌ی بهینه برای جداسازی نمونه‌هایی که در مرحله قبل به آنها اشاره شده است.

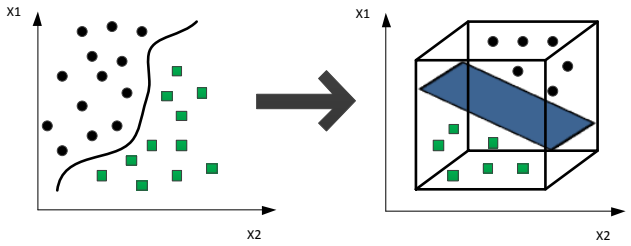
در شکل ۱۰.۲ (آ)، توضیح مصوری از آنچه گفته شده آورده شده است و در شکل ۱۰.۲ (ب) خط جداساز در فضای ورودی (داده‌ها) به صفحه‌ی جداساز در فضای ویژگی‌ها با ابعاد بالاتر تبدیل شده است که به صورت گویاتری این نگاشت را نشان می دهد، اما این حالت بسیار ساده‌ای است و در عمل برای جداسازی ورودی های با ابعاد بالاتر دیگر امکان نشان دادن آنها به صورت آنچه نشان داده شده است امکان پذیر نمی‌باشد.

در نظر بگیرید، اگر فضای داده‌های ورودی ما از ویژگی‌هایی تشکیل شده باشد که بطور خطی از یکدیگر تفکیک پذیر نیستند، تئوری پوشش[۱] بیان می کند برای چنین فضاهای چند بعدی غیرخطی، باید آنها را به فضای ویژگی جدیدی انتقال داد که در آن فضا قابل تفکیک پذیری خطی با احتمال تفکیک بالا، باشند بطوریکه این ۲ موضوع ارضا گردند: اول اینکه، این تبدیل یک تبدیل غیر خطی باشد و دوم اینکه، ابعاد ویژگی‌ها به اندازه کافی بزرگ باشد. این ۲ موضوع در دل شرط ۱ که در بالا ذکر شد گنجانده شده است. اگر x بردارهای ورودی مورد استفاده ما باشند، فرض براین که ابعاد ویژگی آنها m_0 باشد، تابع $\{\phi_i(x)\}_{j=1}^{m1}$ به تبدیل غیرخطی اشاره می کند که فضای بردار ویژگی های ورودی با ابعاد m_0 را به یک فضای برداری دیگر با ابعاد ویژگی m_1 تبدیل می کند. برای تفکیک بردارهای تبدیل یافته جدید ابر صفحه‌ای که پیشنهاد

[۱] Cover's theorem



(آ) نگاشت فضای ورودی به فضای ویژگی



(ب) نگاشت خط جداساز ۲ بعدی به صفحه‌ی مجزاساز

شکل ۱۰.۲: در شکل فوق نگاشتی از فضای ورودی به فضای ویژگی با ابعاد بالاتر را ملاحظه می‌کنید

می‌گردد به اینصورت خواهد بود:

$$\sum_{j=1}^{m1} w_j \phi_j(x) + b = 0 \qquad (۵۱.۲)$$

که در آن $\{w_j(x)\}_{j=1}^{m1}$ بردار وزن هایی است که فضای ویژگی را به فضای خروجی مرتبط می سازد و عبارت b نیز بایاس می‌باشد. خلاصه شده‌ی رابطه‌ی فوق چنین می باشد:

$$\sum_{j=0}^{m1} w_j \phi_j(x) + b = 0 \qquad (۵۲.۲)$$

که در واقع اینطور فرض شده است که $\phi_0(x) = 1$ برای همه‌ی بردارهای x ورودی، بنابراین عبارت بایاس b همان w_0 می باشد. کمیت $\phi_j(x)$ به بردارهای ورودی اشاره می کند که در فضای ویژگی جدید در بردار وزن ضرب خواهند شد تا ابر صفحه‌ی جداکننده را بوجود آورند. در زیر بردار $\phi(x)$ را ملاحظه می‌کنید:

$$\phi(x) = [\phi_0(x), \phi_1(x), \ldots, \phi_{m1}(x)]^T \qquad (۵۳.۲)$$

که در آن $\phi_0(x) = 1$ به ازای تمامی x ها تعریف شده است.

در واقع $\phi(x)$ یک تصویری از فضای ویژگی است که به وسیله‌ی بردارهای ورودی x بوجود آمده است. همانطور که در شکل ۱۰.۲ نیز نشان داده شده است.

حال اگر بخواهیم روابط را به صورت برداری نشان دهیم، می‌توان رابطه‌ی ۵۲.۲ را به صورت برداری چنین نوشت:

$$\boldsymbol{w^T \phi(x) = 0} \tag{۵۴.۲}$$

در رابطه‌ی ۲۹.۲ مقدار بردار وزن را محاسبه نمودیم، حال اگر بخواهیم رابطه‌ی وزن های نگاشت یافته را بوسیله‌ی $\phi(x)$ نمایش دهیم می‌توان نوشت:

$$w = \sum_{j=1}^{N} \alpha_i y_i \phi(x_i) \tag{۵۵.۲}$$

که در آن $\phi(x_i)$ نشان دهنده‌ی الگوی نگاشت شده به ازای ورودی i اُم می باشد. بنابراین اگر رابطه‌ی ۵۴.۲ را در رابطه‌ی ۵۳.۲ جایگذاری نماییم می توانیم صفحه‌ی جداکننده را در فضای ویژگی را چنین بدست آورد:

$$\sum_{j=1}^{N} \alpha_i y_i \phi^T(x_i)\phi(x) = 0 \tag{۵۶.۲}$$

که در آن $\boldsymbol{\phi^T(x_i)\phi(x)}$ یک ضرب داخلی بین ۲ برداری است که در فضای ویژگی وجود دارند، یکی نگاشت شده‌ی بردارهای ورودی x توسط تابع $\phi(x)$ و دیگری هم نگاشت شده‌ی نمونه‌ی i اُم در ورودی‌ها توسط تابع $\phi(x)$، اگر این ضرب داخلی بردارها را با $\mathcal{K}(x, x_i)$ نمایش دهیم و آن را **کرنل** بردارهای x و x_i بنامیم می‌توان نوشت:

$$\begin{aligned} \mathcal{K}(x, x_j) &= \phi^T(x)\phi(x_i) \\ &= \sum_{j=0}^{m1} \phi_j^T(x)\phi_j(x_j) \qquad for\ j = 1, 2, \ldots, N \end{aligned} \tag{۵۷.۲}$$

۱.۲.۲ توابع کرنل

همانطور که در بخش قبل بیان شد، برای یافتن حل مسئله‌ی غیر خطی راه حل حیله‌ی کرنل را معرفی نمودیم، اما در این بخش به معرفی توابعی می پردازیم که شرایطی که در بخش قبل ذکر شد را ارضا کنند یعنی هم غیرخطی بوده و هم یک نگاشت از ورودی‌ها به فضای ویژگی‌ها با ابعاد بالاتر را صورت دهند. در جدول زیر

نمونه‌ای از معروف‌ترین توابع کرنل را ملاحظه می نمایید که بسته به شرایط و کاربرد مختلف مورد استفاده قرار می گیرند.

جدول ۱.۲: توابع کرنل و مشخصات آنها

نوع کرنل	مشخصات کرنل
خطی	xx_i^T
چند جمله‌ای	$(1+xx_i^T)^n$
گاوسی	$e^{-\gamma\|x-x_i\|}$
سخت افزار پسند	$2^{-\gamma\|x-x_i\|}$

نکته‌ای که باید در مورد توابع کرنل و نوع استفاده از آنها در نظر داشت این است که همیشه دقیقاً از قبل مشخص نیست که کدام تابع کرنل برای چه کاربردی باید انتخاب شود و پارامترهای توابع کرنل چه مقدار باید باشد. به عنوان مثال در کاربردهایی که نیاز است تا در مصرف منابع سخت افزاری کاملاً صرفه جویی به عمل آید کرنل سخت افزار پسند به دلیل نداشتن عامل نمایی به شدت در مصرف منابع سخت افزاری یاری می دهد ضمن اینکه نتیجه‌ی آن با نتیجه‌ای که تابع گاوسی می‌دهد قابل برابری است.
برای حل نهایی مسئله از روابط قبل استفاده خواهیم نمود با این تغییر که در روابط جدید باید نگاشتی را که از آن صحبت کردیم به وسیله‌ی توابع کرنل ایجاد گردد. اگر رابطه‌ی ۴۰.۲ را دوباره نویسی کنیم و بجای ضرب بردارهای ورودی، نگاشت شده‌ی آنها را توسط توابع کرنل قرار دهیم، داریم:

$$\begin{aligned} &\min \quad \tfrac{1}{2}\textstyle\sum_i\sum_j \alpha_i\alpha_j y_i y_j \boldsymbol{\phi^T(x_i)\phi(x_j)} - \sum_i \alpha_i \\ &\textit{Subject to}: \\ &\textstyle\sum_i \alpha_i y_i = 0 \\ &0 \le \alpha_i \le C \quad \forall i \end{aligned} \tag{۵۸.۲}$$

که در آن تنها به جای ضرب بردارهای $x_i x_j$ حاصلضرب نگاشت شده‌ی آنها یعنی حاصلضرب $\boldsymbol{\phi^T(x_i)\phi(x_j)}$ قرار داده شده است. می توان به جای حاصلضرب $\boldsymbol{\phi^T(x_i)\phi(x_j)}$ تنها به بیان کرنل مربوطه پرداخت و تنها آن را ذکر کرد در نتیجه شکل کلی معادله‌ی بردار پشتیبان در محیط غیر خطی به شکل زیر خواهد شد:

$$\begin{aligned} &\min \quad \tfrac{1}{2}\textstyle\sum_i\sum_j \alpha_i\alpha_j y_i y_j \mathcal{K}(x_i,x_j) - \sum_i \alpha_i \\ &\textit{Subject to}: \\ &\textstyle\sum_i \alpha_i y_i = 0 \\ &0 \le \alpha_i \le C \quad \forall i \end{aligned} \tag{۵۹.۲}$$

بنابراین معادله‌ی کلی حل کلاسه بندی ماشین‌های بردار پشتیبان در محیط غیر خطی را بدست آوردیم. اگر طبق روابط بخش قبلی بخواهیم ماشین بردار پشتیبان بدست آمده را آزمایش کنیم، طبق رابطه‌ی ۳۰.۲ و با

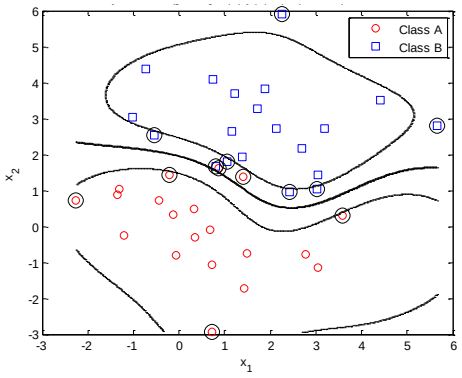
جایگذاری روابط کرنل را در آن می‌توانیم تابع آزمایش را در شرایط غیر خطی بدست آوریم:

$$\begin{aligned} y &= sign(w^T x + b) \\ &= sign(\sum_i \alpha_i y_i x_i^T x + b) \\ &= sign(\sum_i \alpha_i y_i \mathcal{K}(x_i, x) + b) \end{aligned} \tag{۶۰.۲}$$

و همچنین برای محاسبه‌ی پارامتر بایاس نیز می توان از همین رابطه‌ی فوق استفاده کرده و نوشت:

$$b = \frac{1}{|\mathbf{S}|} \sum_i [y_i - \sum_j \alpha_j y_j \mathcal{K}(x_j, x)] \tag{۶۱.۲}$$

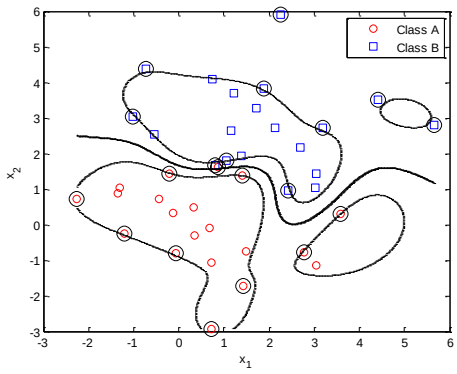
در شکل زیر نمونه‌ای از جداسازی غیرخطی ۲ کلاس به کمک ماشین بردار پشتیبان غیرخطی ملاحظه می کنید:



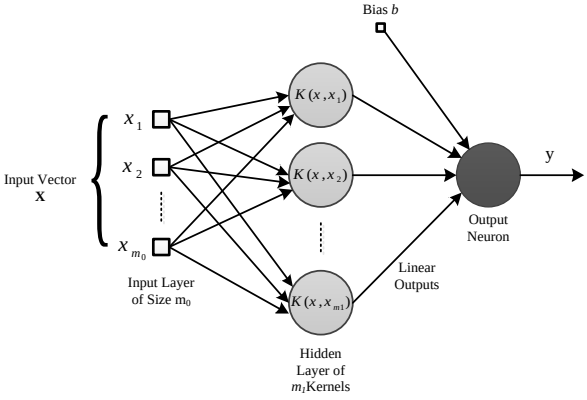
شکل ۱۱.۲: نمونه‌ای از کلاسه‌بندی ماشین بردار پشتیبان با کرنل گاوسی $\sigma = 2$

که در آن پارامتر پراکندگی برای تابع کرنل گوسی زیاد انتخاب شده است. همانطور که ملاحظه می شود حساسیت تابع کرنل به پراکندگی نمونه‌ها کم است اما در شکل زیر این پارامتر کمتر انتخاب شده است که نشان می دهد چگونه نتیجه‌ی تابع گاوسی به این تغییر حتی اندک پارامترهای خود وابسته است.

اگر بخواهیم معماری مطرح شده ماشین بردار پشتیبان را به شکل نرون های شبکه‌های عصبی نمایش دهیم می توان مطابق شکل زیر آن را ترسیم کرد:



شکل ۱۲.۲: نمونه‌ای از کلاسه‌بندی ماشین بردار پشتیبان با کرنل گاوسی $\sigma = 1$



شکل ۱۳.۲: مدل نرون شبکه عصبی ماشین بردار پشتیبان غیرخطی

۲.۲.۲ مثالی عددی از ماشین بردار پشتیبان غیرخطی

برای بیشتر روشن شدن مسئله‌ی ماشین بردار پشتیبان غیرخطی، می خواهیم مثالی عددی را از یک نمونه‌ی غیر خطی ساده بیان کنیم. برای این منظور به بررسی حل تابع XOR خواهیم پرداخت که به کمک ماشین بردار پشتیبان خطی قابل حل نمی باشد. برای حل این مسئله‌ی غیر خطی از کرنل چند جمله‌ای استفاده خواهیم نمود:

$$\mathcal{K}(x, x_i) = (\mathbf{1} + \mathbf{x^T x_i})^{\mathbf{2}}$$

که در آن بردارهای $\mathbf{x_i}$ $\mathbf{x}$ به این ترتیب می باشند:

$$\mathbf{x} = [x_1, x_2]^T$$

$$\mathbf{x_i} = [x_{i1}, x_{i2}]^T$$

بنابراین اگر این ۲ مقدار برای $\mathbf{x_i}$ $\mathbf{x}$ را در رابطه‌ی کرنل جایگذاری نماییم می توان نوشت:

$$\mathcal{K}(\mathbf{x}, \mathbf{x_i}) = [1 + x_1^2 x_{i1}^2 + 2x_1 x_2 x_{i1} x_{i2} + x_2^2 x_{i2}^2 + 2x_1 x_{i1} + 2x_2 x_{i2}]$$

اگر تنها نگاشت بردارهای ورودی $\mathbf{x_i}$ $\mathbf{x}$ را از فضای ورودی به فضای ویژگی را محاسبه نماییم خواهیم داشت:

$$\phi(\mathbf{x}) = [1, x_1^2, \sqrt{2}x_1 x_2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2]$$

$$\phi(\mathbf{x_i}) = [1, x_{i1}^2, \sqrt{2}x_{i1}x_{i2}, x_{i2}^2, \sqrt{2}x_{i1}, \sqrt{2}x_{i2}] \qquad i = 1, 2, 3, 4$$

از آنجاکه تابع کرنل را به ازای ورودی‌ها بسط دادیم می توان از رابطه‌ی زیر استفاده کرده و ماتریس کرنل را محاسبه کنیم:

$$\mathbf{K} = \{\mathcal{K}(x_i, x_j)\}_{(i,j=1)}^{N}$$

که از نتیجه‌ی آن ماتریس K را بصورت زیر خواهیم داشت:

$$\mathbf{K} = \begin{pmatrix} \phi^T(x_1)\phi(x_1) & \dots & \phi^T(x_1)\phi(x_n) \\ \vdots & \ddots & \vdots \\ \phi^T(x_n)\phi(x_n) & \dots & \phi^T(x_n)\phi(x_n) \end{pmatrix} = \begin{pmatrix} 9 & 1 & 1 & 1 \\ 1 & 9 & 1 & 1 \\ 1 & 1 & 9 & 1 \\ 1 & 1 & 1 & 9 \end{pmatrix}$$

از رابطه‌ی ۵۹.۲ و با جایگذاری ماتریس k بدست آمده خواهیم داشت:

$$\begin{aligned} = \alpha_1 \quad & +\alpha_2 + \alpha_3 + \alpha_4 - \tfrac{1}{2}(9\alpha_1^2 - 2\alpha_1\alpha_2 - 2\alpha_1\alpha_3 + 2\alpha_1\alpha_4 \\ & +9\alpha_2^2 - 2\alpha_2\alpha_3 - 2\alpha_2\alpha_4 + 9\alpha_3^2 + 9\alpha_4^2) \end{aligned}$$

اگر بخواهیم مسئله‌ی فوق را برحسب ضرایب لاگرانژ بهینه نماییم به ۴ معدله‌ی زیر خواهیم رسید:

$$9\alpha_1 - \alpha_2 - \alpha_3 + \alpha_4 = 1$$

$$-\alpha_1 + \alpha_2 + \alpha_3 - \alpha_4 = 1$$

$$-\alpha_1 + \alpha_2 + \alpha_3 - \alpha_4 = 1$$

$$\alpha_1 - \alpha_2 - \alpha_3 + 9\alpha_4 = 1$$

۴ معادله‌ی فوق براحتی و با کمی عملیات جبری حل شده و خواهیم داشت:

$$\alpha_{o,1} = \alpha_{o,2} = \alpha_{o,3} = \alpha_{o,4} = \frac{1}{8}$$

از نتایج بدست آمده می‌توان چنین دریافت که تمامی بردارهای ورودی در این مسئله بردارهای پشتیبان می باشند، مقدار بهینه‌ی مسئله‌ی لاگرانژ به ازای مقادیر بدست آمده برای ضرایب α برابر با $\frac{1}{4}$ می باشد. از روابط قبلی می دانیم که معادله‌ی لاگرانژ از رابطه‌ی $\frac{1}{2} \parallel w^T w \parallel$ بدست آمده است، پس مقدار این ۲ باید برابر هم باشد:

$$\frac{1}{2} \parallel w^T w \parallel = \frac{1}{4}$$

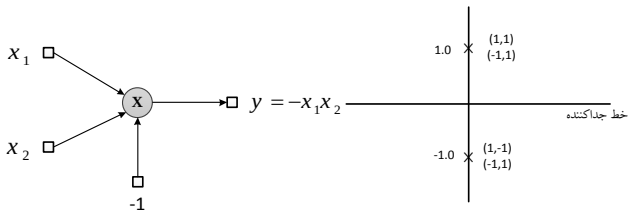
و از آن نتیجه خواهد شد:

$$\parallel w_o \parallel = \frac{1}{\sqrt{2}}$$

همچنین به کمک روابط بدست آمده می‌توانیم مقدار وزن‌های بهینه را نیز محاسبه نماییم:

$$\begin{aligned}
\mathbf{w_o} &= \frac{1}{8}[-\phi_{x1} + \phi_{x2} + \phi_{x3} - \phi_{x4}] \\
&= \frac{1}{8}\left[-\begin{bmatrix} 1 \\ 1 \\ \sqrt{2} \\ 1 \\ -\sqrt{2} \\ -\sqrt{2} \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ \sqrt{2} \\ 1 \\ -\sqrt{2} \\ \sqrt{2} \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ -\sqrt{2} \\ 1 \\ \sqrt{2} \\ -\sqrt{2} \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \\ \sqrt{2} \\ 1 \\ \sqrt{2} \\ \sqrt{2} \end{bmatrix} \right] \\
&= \begin{bmatrix} 0 \\ 0 \\ -\frac{1}{\sqrt{2}} \\ 0 \\ 0 \\ 0 \end{bmatrix}
\end{aligned}$$

همانطور که در شکل زیر هم ملاحظه می‌کنید وضعیت نرون xor و خط جداساز را ملاحظه می‌کنید.



شکل ۱۴.۲: وضعیت مسئله‌ی xor و خط جداساز رسم شده

۳.۲ ماشین‌های بردار پشتیبان چند کلاسه

ماشین بردار پشتیبان در اساس یک کلاسه‌بند باینری(دوتایی) می‌باشد، اما می‌توان با روش‌های مختلفی از این کلاسه‌بند برای حل مسائل چند کلاسه نیز استفاده نمود. چهار روش عمده برای کلاسه‌بندی ماشین بردار پشتیبان چند کلاسه وجود دارد که عبارتنداز:

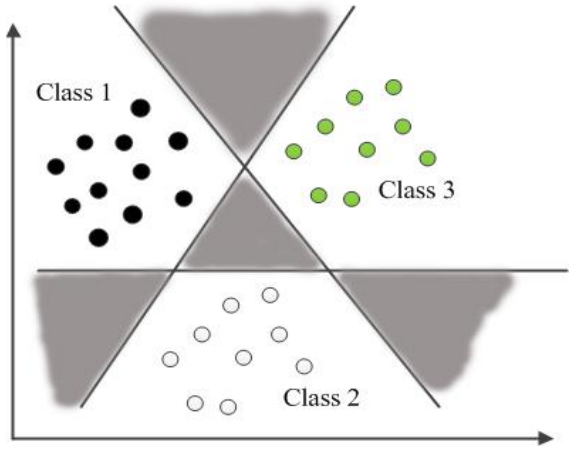
۱. روش یک کلاس در برابر همه

۲. روش کلاسه بندهای دوتایی

۳. روش کد خروجی تصحیح کننده‌ی خطا

۴. روش کلاسه بندی همه کلاس‌ها به یک مرتبه

در روش اول، یعنی روش یک کلاس در برابر همه، اگر n کلاس دادشته باشیم، n تا کلاسه‌بند ماشین بردار پشتیبان دوتایی تشکیل خواهد شد به طوری که برای کلاسه‌بند دوتایی i اُم، کلاس i از بقیه کلاس ها جدا خواهد شد. اما با این روش اگر از توابع تصمیم گسسته استفاده کنیم، نواحی طبقه‌بندی نشده‌ای مطابق شکل زیر بوجود خواهد آمد. برای حل این مشکل شخصی بنام کربل [۱] روش کلاسه‌بندهای دوتایی را مطرح کرد

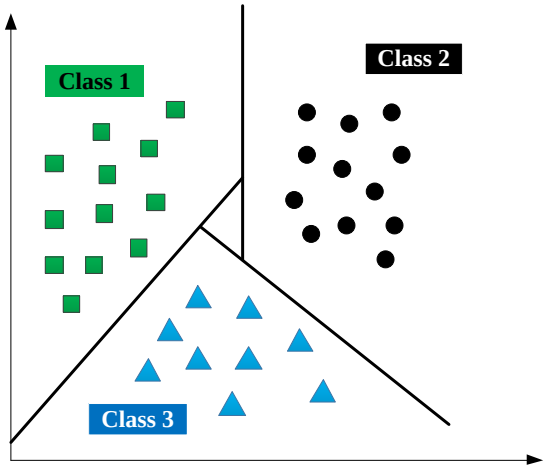


شکل ۱۵.۲: نواحی کلاسه‌بندی نشده در حالت یک کلاس در برابر همه

که در آن برای حل مسئله کلاسه‌بندی n کلاس، از $n(n-1)/2$ مسئله کلاسه‌بندی دوتایی استفاده می‌شود.

[۱] Kerbel

این روش نسبت به روش یک کلاس در برابر همه روش بهتری است اما بازهم نواحی طبقه‌بندی نشده‌ای طبق شکل زیر وجود خواهد داشت. البته نواحی کلاسه‌بندی نشده با استفاده از یکی از روش‌های توابع



شکل ۱۶.۲: کمتر شدن نواحی کلاسه‌بندی نشده به کمک روش کربل

عضویت، درخت‌های تصمیم، روش کد خروجی تصحیح کننده خطا و یا روش همه کلاس‌ها در یکبار اصلاح خواهد شد.

در این نوشتار، از روش کلاسه‌بندی‌های ماشین بردار پشتیبان باینری استفاده خواهد شد. در روش کلاسه‌بندی‌های باینری، در مرحله‌ی آموزش آن، برای همه ترکیب‌های باینری از کلاس‌های i و j ، $i \neq j$مقادیر w_{ij} و مقدار b_{ij} متناظر استخراج شده و برای برای تشکیل توابع تصمیم باینری $d_{ij}(x)$ مورد استفاده قرار می‌گیرد. در هر کلاسه‌بند باینری داده‌های آموزشی کلاس‌های متناظر در تشکیل تابع تصمیم بین آن دو کلاس مورد استفاده قرار می‌گیرد، بنابراین تعداد داده‌های آموزشی نسبت به روش یک کلاس در برابر همه کلاس‌ها، که از کل داده‌های آموزشی برای هر کلاسه‌بند استفاده می‌کند، بسیار کمتر خواهد شد. اما تعداد توابع تصمیم در روش کلاسه‌بندهای باینری، که برابر با $n(n-1)/2$ می باشد، در مقایسه با روش یک کلاس در برابر همه، که برابر n می‌باشد، بیشتر است. در نهایت برای کلاسه‌بندی ورودی x، در روش کلاسه‌بندهای باینری، در هر تابع تصمیم باینری $d_{ij}(x)$ کلاس متناظر استخراج می‌شود و در نهایت کلاسی که بیشترین رأی را داشته باشد، به عنوان کلاس برنده، انتخاب خواهد شد[۵] و [۶] .

فصل ۳

مروری بر پژوهش های پیشین

در این فصل قصد داریم تا برخی مقالات معتبری را که در رابطه با موضوع کلی این پایان نامه می‌باشند، توضیح و تشریح شوند و روش‌هایی را که نویسندگان این مقالات بکار برده‌اند را مورد بررسی قرار خواهیم داد.

۱.۳ ماشین‌های بردار پشتیبان سخت افزار پسند برای کاربردهای خودکار

دیوید آنگویتا[۱] در[۷]، یک نسخه‌ای از ماشین‌های بردار پشتیبان را ارائه می‌دهد، که برای پیاده‌سازی بر روی دستگاه‌هایی که در آنها منابع بسیار محدود می‌باشد، همانند FPGA ها و یا میکروپروسسورها از آنجا که معماری رقم اعشاری هم به ندرت در این دستگاه‌ها یافت می‌شود، بسیار مناسب می‌باشد. نتیجه‌ی آن بر روی یک نسخه‌ی بازاری از مجموعه داده‌های بینایی ماشین برای کاربردهای خودکار(همانند مانیتورینگ راهنمایی و رانندگی) پیاده‌سازی و آزمایش می شود.

مهمترین مسئله‌ای که در مرحله‌ی آزمایش SVM ، بر روی میکروپروسسورهای با محدودیت منابع همانند FPGA ها با آن روبرو هستیم شامل مشکلات زیر می باشد:

۱. طول رجیسترهایی که برای ذخیره‌ی داده‌های مرحله‌ی آموزش بکار می‌رود، کوتاه است.

۲. محاسبات پیچیده ریاضی نیاز است تا بتوان کرنل های مربوطه به SVM را پیاده سازی کرد

می دانیم که معمولاً محاسبات مربوط به آموزش، بصورت خارج از خط و توسط نرم افزارهای محاسباتی صورت می‌گیرد و نتیجه‌ی آن برای پیاده‌سازی بر روی سخت افزار مورد استفاده قرار خواهد گرفت. اگر همه‌ی متغییرهای پیاده‌سازی SVM ، با ممیز شناور صورت گیرد، هزینه‌ی بسیار بالایی را در مصرف منابع

[۱] David Anguita

FPGA متقبل خواهیم شد و همچنین عملیات بسیار سنگینی را که در محاسبه‌ی کرنل‌های مختلف باید صورت گیرد را می توان راحت‌تر و با دقت بسیار خوبی انجام داده و پیاده‌سازی کرد.

در این نوشتار، نویسنده، به کمک تغییرات جزئی در پارامترهای حل مسئله، روش متفاوتی را ارائه می‌دهد. برای یافتن بهترین پارامترهای ممکن، نویسنده ابتدا نرمالیزاسیون زیر را انجام می‌دهد: $\beta_i = \alpha_i \frac{2^n - 1}{C}$ و اگر این نرمالیزاسیون را در رابطه‌ی ماشین بردار پشتیبان که در فصل قبل مشاهده نمودیم، قرار دهیم خواهیم داشت:

$$\min \quad \frac{1}{2}\beta^T K \beta + S^T \beta \qquad 0 \leq \beta_i \leq 2^n - 1 \quad \forall i \in [1, \ldots, \ell]$$

که در آن $S = -\frac{2^n - 1}{C}$ و K کرنل مربوطه می‌باشد. توجه کنید که نتیجه‌ی این رابطه همان نتیجه‌ی مسئله‌ی اصلی را می‌دهد با این تفاوت که در شرایط محدود کننده‌ی آن C حذف شده است و این امکان را می‌دهد تا ضرایب خروجی SVM را دقیقاً با n بیت مدل کرد و تابع آزمایش را هم بصورت زیر می‌توان نوشت:

$$f(x) = \sum_{i=1}^{\ell} \beta_i y_i K(x_i, x)$$

که در آن y_i برچسب‌های کلاس‌های مسئله می‌باشد و نرمالیزاسیون اعمال شده تغییری بر روی این برچسب ها نخواهد داشت و کرنل مورد استفاده توسط نویسنده نیز بصورت زیر می‌باشد:

$$K(x_i, x) = 2^{-\gamma \|x_i - x\|}$$

برای باز هم ساده‌تر کردن پیاده‌سازی، مقدار گاما که در تابع کرنل مشاهده می‌شود را به صورت $\gamma = 2^p$ تعریف کرد که در آن p یک عدد صحیح مثبت و یا منفی می‌تواند باشد. مقادیر تولید شده توسط کرنل را با u بیت نمایش داده شده است که در آن $u \geq 1$ است داریم:

$$0 \leq K(x_i, x) \leq 1 - 2^{-u} \quad \forall i \in [1, \ell]$$

باید توجه داشت که مقدار $K(x_i, x) = 1$ را باید به طور جداگانه‌ای در نظر گرفت. همینطور اگر مقادیر ورودی x را بصورت v بیت در نظر بگیریم که $v \geq 1$ داریم:

$$0 \leq x_i \leq 1 - 2^{-v} \quad \forall i \in [1, m]$$

که در آن m ابعاد بردار ورودی x است.

در ادامه‌ی این پژوهش، مقادیر مطلوب را برای بازه‌های کرنل و ورودی‌ها را با استفاده از روش بازه‌بندی ریاضیاتی پیدا می کند که بطور ساده‌ای فضای جستجو را به ۲ قسمت حد بالای بازه و حد پایین بازه محدود می‌کند که می‌توان بصورت بازه‌ی $[lowerbound, lowerbound]$ آن را نشان داد. بنابراین طبق این چهارچوب می‌توان نوشت:

$$k_i \subseteq [0, 1 - 2^{-u}]$$

$$x_i \subseteq [0, 1 - 2^{-v}]$$

که در آن K_i کرنل مربوط و x_i بردار ورودی می‌باشد.

دیوید آنگوتا، روش خود را بر روی سیستم کاربردی خودکار آزمایش نمود. در اینجا کاربرد شناسایی افراد عابر پیاده برای کاربردهای مانیتورینگ استفاده شده است که برای انجام آن از مجموعه داده‌های معتبر و تجاری استفاده شده است. این مجموعه داده‌ها شامل تصاویر سیاه و سفید ۸ بیتی می‌باشد. لازم به ذکر است برای کابردهای بلادرنگ و آنی می بایست مرحله‌ی آموزش بر روی مجموعه داده‌ها بطوری انجام گردد که در آن تعداد کمتری از داده‌ها برای آموزش و تعداد بسیار بیشتری برای آزمایش انتخاب گردند این در حالی است که برای سایر کاربردهای شبکه عصبی این رابطه برعکس می‌باشد. در این مقاله $\frac{1}{7}$ داده‌ها برای آموزش و $\frac{7}{8}$ داده‌ها نیز برای مرحله‌ی آزمایش در نظر گرفته شده‌است. تعداد بیت هایی که متغییرهایی که SVM را تشکیل می‌دهند می‌توانند مقادیر متفاوتی را بپذیرند.

۱. n : تعداد بیت‌های مورد استفاده برای β

۲. u : تعداد بیت‌های مورد استفاده برای کرنل مورد استفاده

در جدول ۱.۴ مقادیر خطا در حالت استفاده از ممیز شناور و در جدول ۲.۴ مقدار خطای بدست آمده به مورد استفاده از ۸ بیت برای مقدار v که بردار ذخیره کننده‌ی بردار ورودی بود، نشان داده شده است.

جدول ۱.۳: مقدار خطای محاسبه شده برای آزمایش ماشین بردار پشتیبان به کمک روش ممیز ثابت

(%) rate Error	γ	**C**
5.96	2^{-7}	10

جدول ۲.۳: خطای محاسبه شده برای آزمایش SVM به کمک ۸ بیت اختصاص داده شده برای ذخیره‌ی ورودی‌ها

16	12	8	$n = 6$	
9.29	9.28	10.54	14.58	$u = 6$
6.19	6.18	9.83	17.66	8
5.96	5.96	11.38	22.97	12
5.96	5.96	10.53	20.58	16

همانگونه در جدول فوق نیز مشاهده می‌شود، به ازای مقدار ۱۲ و ۱۶ بیت برای به ترتیب مقادیر کرنل و خروجی مسئله و به ازای مقدار ۸ بیت برای ورودی‌ها به همان میزان دقتی در پیاده‌سازی ماشین بردار پشتیبان می‌رسیم که با روش ممیز شناور به آن رسیده بودیم و این نشان می‌دهد می‌توان بدون استفاده از روش ممیز شناور و تنها با اختصاص دادن تعداد بیت‌های مناسب و روش معرفی شده به همان دقت دست یافت.

۲.۳ پیاده‌سازی کلاسه‌بند ماشین بردار پشتیبان برروی FPGA

در مرجع[۸] آقای رویز لاتا[۱] طراحی موفقیت آمیزی از ماشین بردار پشتیبان را برای کلاسه‌بندی و رگرسیون توابع با عملکرد بسیار بالا و با مصرف کم حافظه را معرفی می‌کند و سپس سیستم معرفی شده را بر روی یک مدل ارزان قیمت از FPGA ها پیاده‌سازی می‌کند. یکی از مشکلاتی که بر سر راه فراهم کردن یک عملیات محاسباتی پیچیده برای چنین سیستمی وجود دارد فراهم آوردن احتیاجات مسئله می‌باشد:

۱ منابع سخت افزاری بسیار محدود

۲ قابلیت یادگیری از روی نمونه‌ها

بنابراین این مقاله بر روی سخت افزاری تمرکز می‌کند که هم از لحاظ منابع پروسسوری بسیار مناسب باشد و هم بتواند عملیات محاسباتی پیچیده را به خوبی انجام دهد، لذا در اینجا یک نسخه از FPGA های خانواده Xilinx مورد استفاده قرار می‌گیرد.

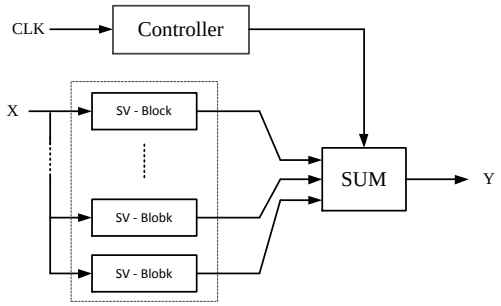
پردازنده‌های عصبی، برای کاربردهای بینایی ماشین و سایر کاربردها تعبیه شده‌اند. این دستگاه‌ها مزایای زیادی از قبیل قابلیت برنامه‌ریزی سریع، انعطاف پذیری بالا، زمان توسعه بسیار سریع بدون افت کارآیی مرتبط با طراحی را دارا می‌باشد. علاوه بر این پردازش موازی هم محقق می‌شود، که یک مزیت مهم نسبت به سایر دستگاه‌ها و ادوات دیجیتالی همانند DSP ها و سایر میکروپروسسورها می‌باشد. در بحث سخت‌افزاری طراحی انجام شده در این مقاله مرحله‌ی آزمایش معماری زیر پیشنهاد شده است:

همانگونه که در شکل ۱.۳هم ملاحظه می‌شود هر بلوک SV مرتبط با هر بردار پشتیبان می‌باشد، ورودی x و خروجی مرتبط با هر کدام از قسمت‌های کلاسه‌بندی توسط کنترل کننده‌ای که با کلاک هماهنگ است، کنترل می‌گردد. معماری هر کدام از SV-Block ها در شکل زیر نشان داده شده است:

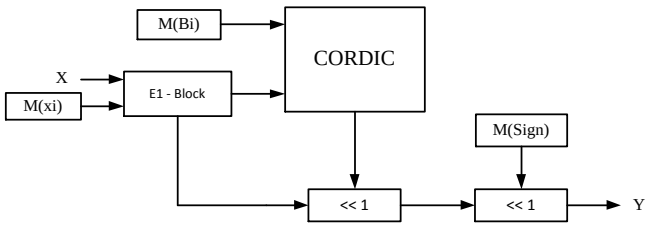
معماری داخلی هر بلوک SV-Block از بلوک‌های حافظه (که با حرف M مشخص شده است)، بلوک E_i که نشان دهنده‌ی رابطه‌ی $-\gamma \parallel x_i - x \parallel$ می‌باشد که توسط بلوک محاسباتی CORDIC [۲] می‌گردد، همچنین

[۱] Marta Ruiz-L lata

[۲] **CO**ordinate **R**otation **DI**gital **C**omputer



شکل ۱.۳: معماری انجام شده برای مرحله‌ی آزمایش توسط مقاله



شکل ۲.۳: طراحی داخلی بلوک‌های بردار پشتیبان در معماری معرفی شده

کلاک و شیفت رجیسترها تشکیل یافته است. عملکرد معماری طراحی شده برای یک مسئله کلاسه‌بندی بر روی مجموعه داده‌ی COIL مورد آزمایش قرار گرفته است. این مجموعه داده نیز از تصاویر ۸ بیتی سیاه و سفید دارای کیفیت ۳۲×۳۲ تشکیل شده است. در این پیاده‌سازی به دلیل وجود چند کلاس، از استراتژی خاصی استفاده شده است که به $q-1$ کلاسه بند برای کلاسه‌بندی q کلاس نیاز باشد در صورتیکه در حالت یک کلاس در برابر همه، به تعداد $q(q-1)/2$ کلاسه‌بند برای q کلاس نیاز بود.

۳.۳ ماشین‌های بردار پشتیبان موازی PSVM در مرحله‌ی آموزش

در برخی مقالات مجله و کنفرانس‌های معتبر برای بهبود عملکرد ماشین‌های بردار پشتیبان در مرحله‌ی آموزش الگوریتم‌ها و پیشنهادهایی آورده شده است که بیشتر هدف آنها بهینه‌سازی ماشین‌های بردار پشتیبان از لحاظ مقدار منابع مصرفی جهت پیاده‌سازی و همچنین سرعت انجام محاسبات بوده است و دقت عملکرد و صحت نتایج بدست آمده را بررسی نمی کنند زیرا تنها این الگوریتم‌ها بر قسمت‌هایی از تئوری ماشین بردار پشتیبان وارد می‌شوند که تأثیری بر روی دقت عملکرد سیستم ندارد.

ادوارد چانگ در مقاله‌ی خود آنگونه توضیح داده است که ماشین‌های بردار پشتیبان از لحاظ مقدار زمان مورد نیاز برای ارائه‌ی نتایج و همچنین مقدار حافظه‌ی مصرفی نیاز به بهینه‌سازی دارند. برای بهینه کردن این موارد ایشان الگوریتمی به نام ماشین‌های بردار پشتیبان موازی PSVM را ارائه داده اند مقدار حافظه‌ی مصرفی را بر اساس این الگوریتم می توان به مقدار قابل توجهی کاهش داد[۹]. این الگوریتم از طریق پایگاه اینترنتی http://code.google.com/p/psvm قابل دسترسی می‌باشد.

در مرجع[۱۰] آقای ژوانگ چاوو[۱] یک پیاده‌سازی بهینه و جدیدی را برای ماشین بردار پشتیبان موازی ارائه کرده‌اند که می‌تواند برای کارکردن با مجموعه داده‌های بسیار بزرگ، بسیار مؤثر واقع شود. این الگوریتم جدید برای PSVM اغلب برای کاهش حافظه‌ی مصرفی و صرفه‌جویی در زمان مرحله‌ی آموزش کارآمد است و بر اساس همین ۲ فاکتور با بقیه مقالات مقایسه می‌گردد. سیستم ارائه شده توسط این مقاله به میزان ۴ برابر سریعتر از نرم افزار معروف LibSVM عمل کرده و برای پیاده‌سازی برای هسته‌های دیجیتالی به میزان بسیار قابل توجهی در مصرف منابع سخت افزاری بهینه عمل خواهد کرد. اساس این الگوریتم در مرحله‌ی آموزش تجزیه‌ی ماتریس‌های مرحله‌ی آموزش و سپس به کار بردن آنها است. به عنوان مثال فرض کنید، ماتریس‌های α_i ، y_i و Q را بصورت زیر تجزیه می‌کنیم.

$$\alpha_i = \begin{bmatrix} \alpha_\beta \\ \alpha_N \end{bmatrix} \quad , y_i = \begin{bmatrix} y_\beta \\ y_N \end{bmatrix} \quad , Q = \begin{bmatrix} Q_{\beta\beta} & Q_{\beta N} \\ Q_{N\beta} & Q_{NN} \end{bmatrix}$$

که در آنها β به قسمتی از مجموعه داده‌های به تعداد n متغییر اشاره دارد و N به قسمتی از مجموعه داده‌ها به تعداد $\ell - N$ اشاره دارد. سپس با جایگذاری این ماتریس‌های جایگزین در روابط اصلی و استفاده از روش SMO[۲] برای حل کردن معادلات غیرخطی بوجود آمده این الگوریتم به مورد اجرا در می‌آید.

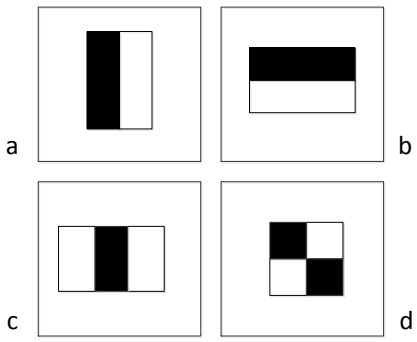
در سایر مقالات از جمله [۱۱] ،[۱۲] و[۱۳] موضوعاتی مشابه، در مورد بهینه کردن فاز آموزش ماشین بردار پشتیبان و پیاده سازی آن بر روی میکروپروسسورها مورد بحث و بررسی قرار گرفته است.

[۱] xuang chau
[۲] Sequential Minimal Optimization

۴.۳ شناسایی چهره و پیاده‌سازی آن به کمک ماشین‌های بردار پشتیبان

در این مقاله، یک ترکیب ۳ مرحله‌ای برای افزایش سرعت سیستم شناسایی چهره به کمک ماشین بردار پشتیبان ارائه خواهد شد. در این سیستم پیشنهاد شده، مقدار بسیار زیادی از الگوهایی که غیر چهره می‌باشند به سرعت و به کمک کلاسه بند AdaBoost از تصویر مورد آزمایش کنار گذاشته خواهند شد و در مرحله‌ی آخر به کمک SVM و با دقت شناسایی چهره مورد بررسی قرار می‌گیرد. در این سیستم در تمامی مراحل از ویژگی شبه هار[۱] استفاده شده است و انتخاب ویژگی‌ها به کمک کلاسه‌بند AdaBoost صورت می‌گیرد که هم سرعت و هم دقت کار را تا حد خوبی بالا می‌برد.

در مورد ویژگی‌های هار باید خاطر نشان کرد که این ویژگی‌ها به دلیل سادگی و قابلیت انعطاف خوبی که از خود نشان می‌دهند. همچنین دارای پیچیدگی محاسباتی کمتری می‌باشند مورد استفاده قرار می‌گیرند که اساس و کار آنها در [۱۴] آورده شده است. به طور مشخص در سیستم طراحی شده در این مقاله از ۳ نوع ویژگی هار استفاده گردیده است که به کمک مستطیل‌هایی با سایزهای مختلف محاسبه می‌شوند. این مستطیل‌ها را در شکل۳.۳ ملاحظه می‌کنید.

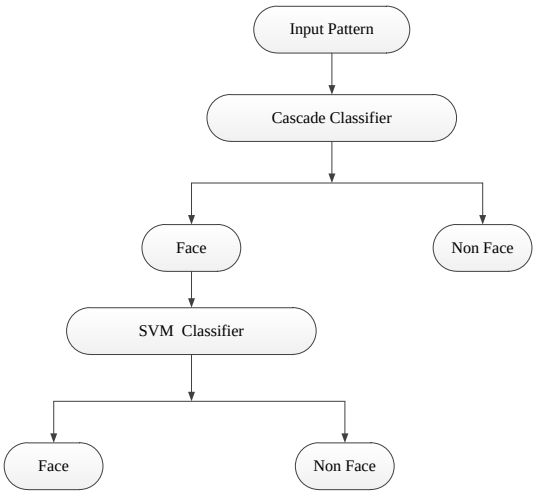


شکل ۳.۳: محاسبه‌ی ویژگی به کمک مستطیل‌های هار

۲ مستطیل a و b ویژگی‌های لبه را در ۲ حالت افقی و عمودی استخراج می‌کنند، حالت c ویژگی خط را و حالت d ویژگی خطوط قطری را محاسبه می‌کنند. مجموع پیکسل‌های تصویر در قسمت تیره رنگ از قسمت سفید کسر می گردد و این مقدار به عنوان یک ویژگی محسوب می‌گردد. می توان مستطیل‌هایی با طول و عرض مختلف تولید کرده و در سرتاسر تصویر ویژگی‌های مختلف را به این روش استخراج کرد. این ویژگی‌ها به سرعت و سادگی مشهورند. توضیحات کامل‌تر در این مورد را می‌توانید در [۱۵] ببینید.

[۱] Haar Like features

در مرحله‌ی آموزش از ۵۰۰۰ چهره در اندازه‌ي ۲۴×۲۴ استفاده شده است در حالیکه مجموعه داده‌های غیرچهره از ۷۰۰ تصویر تشکیل شده است. همانطور که می‌دانید کلاسه‌بند AdaBoost می‌تواند از تعداد قابل کنترلی $WeakClassifier$ و همچنین $StrongClassifier$ بهره ببرد لذا این مسئله در دست طراح است که طوری کلاسه‌بند خود را طراحی کند که هم در سرعت و هم در دقت بیشترین ملاحظه به عمل آید زیرا همواره یک معاوضه بین دقت و سرعت در این سیستم‌ها وجود دارد. کلاسه‌بند AdaBoost می‌تواند تا دقت 99.99% را در مرحله‌ی آموزش را تحویل دهد اما باید در سرعت انجام محاسبات هم دقت و هم صرفه‌جویی به عمل آید. در شکل ۴.۳ روند شناسایی چهره به کمک کلاسه‌بندهای AdaBoost و SVM را ملاحظه می نمایید.



شکل ۴.۳: روند شناسایی چهره به کمک کلاسه‌بندهای AdaBoost و SVM

طراح در این مقاله در مرحله‌ی آموزش به طوری که توضیح داده شد دقت 99.89% را بدست آورده است در حالیکه در آزمایش سیستم طراحی شده به کمک مجموعه داده معتبر $MIT+CMU$ دقت 68% را در شناسایی چهره کسب کرده است.

۵.۳ پیاده‌سازی ماشین بردار پشتیبان بر روی FPGA

در مرجع[۱۶]، نویسنده، نتایج یک پیاده‌سازی سخت افزاری پیکره‌بندی شده توسط کلاسه‌بند ماشین بردار پشتیبان را ارائه می‌کند، که برای کاربردهای هوش مصنوعی که در آنها نیاز به تصمیمات آنی و بلادرنگ دارند ازجمله در وسایل و ادوات قابل حمل، مورد استفاده است.

طراحی ماشین بردار پشتیبان، نیازمند محاسبات داده‌ای بسیار زیاد و نیازمند حل عددی یک مسئله درجه دوم و بهینه‌سازی آن می‌باشد که نیازمند محاسبات پیچیده و وقت گیر مخصوصاً روابط پیچیده ریاضی در مرحله‌ی آموزش می‌باشد. بنابراین اگر بخواهیم کاربرد خاصی را به کمک SVM عملی سازیم، به دلیل محاسبات زیاد در مرحله‌ی آموزش کار کمی محدود می‌شود. بنابراین در کاربردهای بلادرنگ نیازمند یک پشتیبانی قوی می‌باشیم که بتوانیم به نحوی تعادلی در این محاسبات پیچیده و زمان زیادی که صرف می‌شود برقرار سازیم. در برخی کاربردهای خاص همانند شناسایی چهره که در آن نیازی نیست تا مراحل آموزش تکرار گردند و تنها یکبار آموزش برای یادگیری کافی می‌باشد. می‌توان مرحله‌ی آموزش ماشین بردار پشتیبان را به صورت جداگانه و بوسیله‌ی یک نرم افزار کامپیوتری همانند MATLAB انجام داده و در یک حافظه ذخیره کنیم و سپس از نتایج آن برای آزمایش کلاسه‌بند خود بر روی سخت افزار استفاده نماییم.

تلاش‌های متفاوتی در مورد طراحی یک ماشین بردار پشتیبان بهینه در مورد معماری مرحله‌ی آموزش و آزمایش صورت گرفته است که برای مثال می‌توان به مقاله‌ی آقای پروفسور آنگوتا اشاره کرد [۱۷]. در منبع دیگری [۱۸] همان نویسنده یک پیاده سازی دیجیتالی برای یادگیری کرنل‌های مختلف توسط ماشین بردار پشتیبان ارائه کرده است.

در این مقاله، نویسنده از سخت افزار $virtexII - xc2v1000$ استفاده کرده است که ادوات جانبی آن بصورت

۱ ۲ عدد بانک $ZBT\ SRAM$

۲ رابط سخت افزاری RS-۲۳۲

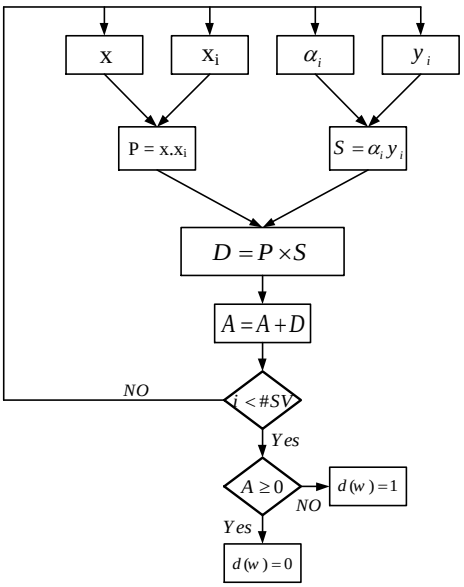
۳ پشتیبانی ویدئو با پهنای باند ۴MB

۴ پورت پیکره‌بندی موازی

می‌باشد و همچنین فرکانس کاری سیستم هم ۵۰MHz می‌باشد. جهت طراحی سخت افزار برای پیاده‌سازی تابع تصمیم می توان قدم به قدم مراحل شکل ۵.۳ را دنبال کرد:

که در آن $P = x_i^T x$ حاصلضرب نقطه‌ای، $S = \alpha_i y_i$ حاصلضرب اسکالر و $D = P \times S$ حاصلضرب ۲ پارامتر P و S و در آخر $A = A + D$ یک جمع کننده می‌باشد که حاصل آن وارد تابع تصمیم نهایی خواهد شد.

حال مراحل مختلف عملکرد پیاده‌سازی بر روی FPGA را بر حسب تابعی از زمان رسم می‌کنیم که در هر زمان بخصوص FPGA در حال انجام چه عملیاتی می‌باشد. اگر بخواهیم این الگوریتم تصمیم‌گیری SVM



شکل ۵.۳: روند طراحی بکار برده شده در این مقاله

را برای ورودی‌های x آزمایش کنیم چهار مرحله‌ی اصلی که در صفحه‌ی قبل گفته شد باید به ازای تمامی بردارهای پشتیبان تکرار گردد. بنابراین ملاحظه می‌شود که اگر تعداد بردارهای پشتیبان بسیار زیاد باشند به تعداد عملیات ضرب و جمع خیلی زیادی نیاز داریم و این یعنی باید واحدهای جداول جستجوی زیادی را از FPGA تقاضا کنیم. در شکل ۶.۳ مراحل مختلف پیاده‌سازی بر روی FPGA را بر اساس زمان ملاحظه می‌نمایید.

در این دیاگرام زمانی مراحل زیر به ترتیب صورت می‌گیرد:

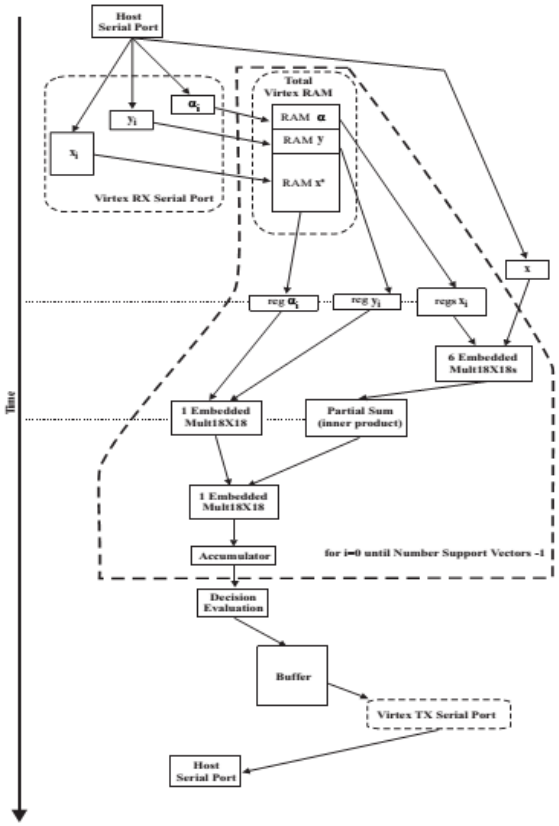
۱ پورت سریال بردارهای α_i، y_i و x_i را وارد می‌کند.

۲ پورت سریال داده‌ها را دریافت می‌کند و آنها را در RAM ذخیره می‌کند.

۳ سخت افزار منتظر ورود بردارهای جدید x است تا بتواند آنها را وارد پروسه‌ی کلاسه‌بندی کند.

۴ وقتی داده‌های x دریافت شد، α_i، y_i و x_i از RAM خوانده شده و به صورت همزمان وارد مرحله‌ی بعد می‌شود.

۵ وقتی خواندن تمام شد، حاصلضرب s و همچنین حاصلضرب برداری x_i و x بصورت همزمان انجام می‌گیرد.



شکل ۶.۳: نمودار عملکرد زمانی FPGA طراحی شده توسط نویسنده مقاله

۶ سرانجام مقادیر حاصلضرب‌ها در بردار P ذخیره می‌شوند.

۷ مقدار حاصلضرب $D = P \times S$ محاسبه می‌شود.

۸ D در داخل A ذخیره می‌شود و با A جمع می‌شود. $A = A + D$

۹ مرحله‌ی ۴ تا ۸ به ازای هر بردار پشتیبان تکرار می‌شود.

۱۰ وقتی این حلقه تمام شد، علامت A محاسبه می‌شود، بخاطر اینکه بتوانیم کلاس مربوط به ورودی‌های x را تشخیص دهیم.

۱۱ نتیجه‌ی کلاسه‌بندی در یک بافر ذخیره می‌شود.

۱۲ سخت افزار منتظر الگوی دیگری برای کلاسه‌بندی است وقتی داده‌های جدید وارد شوند مرحله‌ی ۴ تا ۱۱ دوباره صورت می‌گیرد.

۱۳ وقتی بافر پر شد، سخت افزار به کمک پورت سریال نتایج کلاسه‌بندی را می‌فرستد.

۱۴ host نتایج ارسال شده را دریافت کرده و آنها را در یک فایل ذخیره می‌نماید.

از آنجا که می‌خواهیم عملکرد اجرایی معماری را ببینیم دو نوع SVM خطی طراحی شده است، اولین SVM که بر اساس کلاسه‌بندی و تجزیه کلاس‌های چند بعدی گاوسی که بوسیله‌ی تابع گاوسی با میانگین و واریانس‌های متفاوت ایجاد شده‌اند و دارای ۱۰۰۰ نمونه می‌باشند، آموزش داده شده است و دومین SVM برای کلاسه‌بندی تصاویر MRI گرفته شده که بر اساس سه مدل مختلف هستند، تمامی تصاویر دارای یک اندازه و ابعاد مشخص می‌باشند و همگی دارای ۲۵۶×۲۵۶ می‌باشند. از آنجاکه ساختار SVM طراحی شده برای ویژگی‌های ۶ بعدی طراحی شده‌اند و از آنجاکه تنها سه سطح خاکستری برای هر تصویر در نظر گرفته می‌شود، بقیه‌ی ابعاد ویژگی‌های باقیمانده(۳ تا از ۶ تا) با صفر پر می‌شوند. در هر دو SVM طراحی شده، آموزش حل مسئله درجه دوم بوسیله‌ی نرم افزار LibSVM در محیط MATLAB صورت پذیرفته است. در قسمت آموزش آن پارامترهای SVM بصورت ممیز ثابت محاسبه شده‌اند. برای هر کلاس هم ۱۰۰ نمونه تست در نظر گرفته شده است و در هر کلاس بوسیله‌ی نرم افزار LibSVM داده‌ها بصورت 100% درست کلاسه‌بندی شده‌اند. در قسمت نتایج، در جدول ۳.۳ نحوه‌ی عملکرد SVM اولی آورده شده است که نشان می‌دهد برای داده‌های تست 97% جواب درست حاصل شده است. در مورد SVM دومی مرحله‌ی آموزش بوسیله‌ی ۷۸ بردار پشتیبان صورت پذیرفته است، که نسبت به SVM اولی نیازمند صرف زمان و دقت بیشتری برای محاسبه تک تک بردارهای پشتیبان در الگوریتم آزمایش می‌باشد. نتایج SVM دومی هم در جدول ۳.۶ نشان داده شده است که نشان می‌دهد در مرحله‌ی آزمایش 95% درستی حاصل شده است.

جدول ۳.۳: نتایج پیاده‌سازی SVM اولی بر روی نرم‌افزار و سخت‌افزار

Virtex		MATLAB		
C_1	C_{-1}	C_1	C_{-1}	
49	2	50	0	C_1
1	48	0	50	C_{-1}

جدول ۴.۳: نتایج پیاده‌سازی SVM دومی بر روی نرم‌افزار و سخت‌افزار

Virtex		MATLAB		
C_1	C_{-1}	C_1	C_{-1}	
49	4	50	0	C_1
1	46	0	50	C_{-1}

و در آخر جدول ۵.۳ نشان می‌دهد که اصل هرف این کار به طور موفقیت آمیزی به نتیجه رسیده است. زمان پردازش در برابر مدل نرم‌افزاری به نسبت ۱:۲ می‌باشد. حتی با در نظر گرفتن تغییرات فرکانس کلاک سیستم که در سیستم سخت افزاری $50MHz$ و در مدل نرم افزاری $550MHz$ می‌باشد، سرعت سخت افزاری $\frac{1}{2}$ برابر مدل نرم افزاری شده است (البته با در نظر نگرفتن سرعت کلاک سیستم‌ها).

جدول ۵.۳: گزارش زمانیِ مراحل مختلف پیاده‌سازی

$(550MHz)$PC	$(50MHz)$FPGA	Time
$--$	$63.7\ s$	T_{RxTx}
$59.5\ s$	$173.4\ s$	T_{total}
$--$	$109.7\ s$	T_{FPGA}
$0.91\ ms$	$1.67\ ms$	T_{vector}
$11.64\ \mu s$	$21.46\ \mu s$	T_{sv}

فصل ۴

FPGA وکاربرد آن در پردازش داده

۱.۴ مروری مقدماتی بر بوردهای FPGA

یک بورد FPGA ، مدار مجتمع همه منظوره‌ای است که بیشتر از آنکه کارخانه‌ی سازنده‌ی آن، آن را برنامه‌ریزی کرده باشد، طراحی که قصد استفاده از آن را دارد آن را برنامه‌ریزی می‌کند. برخلاف مدارهای مجتمع ASIC[۱] که می توانند عملکرد مشابهی را در یک سیستم الکترونیکی انجام دهند، FPGA ها می‌توانند حتی بعد از آنکه برای استفاده در یک سیستم خاص آماده شدند، دوباره برنامه‌ریزی شوند در حالیکه ASIC ها تنها برای استفاده در یک برنامه‌ی خاص برنامه‌ریزی و آماده‌سازی می شوند. یک چیپ FPGA از طریق بارگیری یک فایل رشته بیتی[۲]، توسط برنامه‌ی پیکره‌بندی کننده‌ی خاص بر روی حافظه‌ی موقتی که بر روی چیپ FPGA قرار دارد، برنامه‌ریزی می‌شود. کاملاً مشابه کد هدف[۳]، که برای برنامه‌ریزی میکروپروسسورها مورد استفاده قرار می‌گیرد. این رشته بیتی محصولی از ابزارهای کامپایل کننده است که سطوح بالای برنامه که توسط طراح، طراحی شده است را به سطوح پایین تر ولی قابل پیاده‌سازی بر روی ماشین تبدیل می‌کند. نرم افزار *Xilinx System Generator* پیشرو در ارائه‌ی این ایده است که می‌تواند یک مرحله از زبان سطح بالای Simulink را جهت پیاده‌سازی بر روی بورد FPGA مهیا کند. پروسسور FPGA ، یک آرایه‌ی دو بعدی از منابع قابل پیکره‌بندی را مهیا می‌کند که می‌تواند یک گستره‌ی وسیعی از توابع منطقی و ریاضیاتی را پیاده‌سازی کند. منابعی همچون بلوک‌های DSP، ضرب کننده‌ها، حافظه‌های ۲ پورتی، جدول‌های جستجو[۴]، رجیسترها، بافرهای ۳ وضعیتی، مالتی پلکسرها، کلاک دیجیتال و ... در این گستره از منابع اختصاص داده شده‌اند. همچنین این بوردها ورودی-خروجی های پیچیده‌ای را شامل می‌شود که

[۱] Application-Specific Integrated Circuit
[۲] Bitstream
[۳] Object Code
[۴] Look Up Table

می‌تواند رنج وسیعی از پهنای باند برای دیتاهای ورودی و خروجی و ولتاژ ورودی مورد نیاز را پشتیبانی کند. چیپ‌های FPGA ادوات پردازش داده با عملکرد فوق‌العاده بالا می‌باشند. عملکرد DSP پروسسورها به دلیل عملکرد بسیار خوب FPGA ها در توانایی ایجاد معماری پردازش موازی برای پردازش داده‌ها، تحت‌الشعاع قرار گرفته است. در مقایسه با میکروپروسسورها و DSP پروسسورها، که عملکرد آنها به حداکثر نرخ کلاکی که میکروپروسسور توانایی اجرا کردن برنامه‌ها در آن نرخ کلاک را دارد، عملکرد FPGA ها به نحوه‌ی معماری صورت گرفته توسط طراح و الگوریتم محاسباتی پیاده‌سازی شده که تا چه اندازه امکان موازی‌سازی در آن وجود دارد، بستگی خواهد داشت. یک ترکیب مناسب از کلاک‌های با نرخ بسیار بالا (در حال حاضر بوردهایی با نرخ کلاک با فرکانس ۱۰۰تا ۲۰۰ مگاهرتز وجود دارد)، همچنین وجود حافظه‌ی به شدت توزیع شده[۱] می تواند سیستمی را بوجود آورد که توانایی آن را دارد تا بتواند پردازش داده و همچنین سایر برنامه‌ها و الگوریتم‌ها را به خوبی و با سرعت چشمگیری انجام دهد. به عنوان مثال، یک مدل از FPGA با مقدار حافظه‌ی مناسب و با نرخ کلاک ۱۵۰ مگاهرتز می‌توانند صدها ترابایت[۲] داده را در هر ثانیه پردازش کند. از مهمترین کاربردهایی که سیستم‌هایی با چنین ساختار داخلی دارند را می‌توان به کنترل ربات‌ها[۳۴]، پردازش سیگنال دیجیتال[۳۵]، ترتیب‌گذاری رشته‌های DNA [۳۶]، واحدهای محاسباتی کنترلی و ... اشاره کرد.

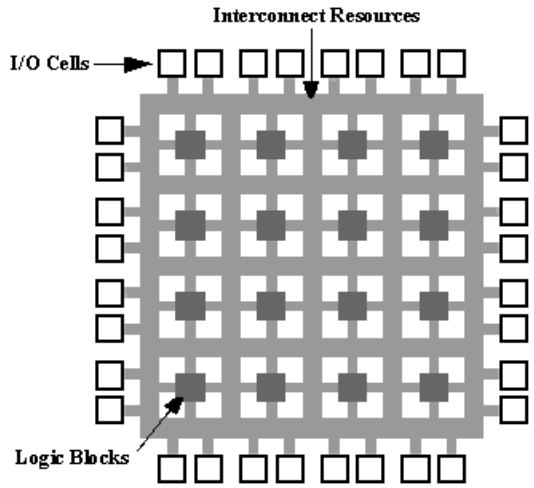
در برخی منابع روش ارائه شده است که بتوان مرحله‌ی آموزش را نیز به کمک سخت‌افزار انجام داد امّا معمولاً برای جلوگیری از محاسبات پیچیده و طولانی و همچنین برای کاربردهایی که نیاز نیست شبکه روی داده‌های آموزش بیشتر از یکبار آموزش داده شوند، مرحله‌ی آموزش را به صورت جداگانه و به کمک یک کامپیوتر و نرم‌افزار مربوط همانند MATLAB استفاده می‌کنیم[۳۷]-[۳۹] .

۲.۴ کاربردهای پردازش داده به کمک FPGA

کاربردهای پردازش داده‌ی بسیار زیادی وجود دارد(به عنوان مثال ساده مبدل‌های دیجیتال)، که می‌توانند بر روی یک مدار مجتمع سفارشی و یا بورد دیجیتال FPGA پیاده‌سازی شود. یکی دیگر از مزایای استفاده از FPGA ها کاهش قابل ملاحظه‌ای در قیمت نهایی آن نسبت به بوردهای سفارشی می‌باشد که این کاهش قیمت به این دلیل می‌باشد که FPGA ها محصولات با تولید انبوه می‌باشد که در هر مرحله از مراحل استفاده در سیستم مورد نظر قابل برنامه ریزی و پیکره‌بندی مجدد می‌باشد در حالیکه میکروپروسسورهای

[۱] Highly-Distributed Memory
[۲] TeraBytes

سفارشی فقط برای یک منظور خاص آماده‌سازی شده‌اند. اگر بخواهیم FPGA ها را به طور ساده تشریح کنیم، عبارتست از یک تراشه که از تعداد بالایی بلوک منطقی ($LogicBlock$)، خطوط ارتباطی و پایه‌های ورودی/خروجی(IOB) تشکیل شده است که به صورت آرایه هایی در کنار یکدیگر قرار دارند. خطوط ارتباطی که وظیفه‌ی آنها ارتباط بین بلوک‌های منطقی است از سوئیچ های قابل برنامه‌ریزی تشکیل شده‌اند. این سوئیچ ها بسته به نوعی که دارند، برخی تنها یکبار قابل برنامه‌ریزی هستند و برخی به تعداد دفعات زیادی برنامه‌ریزی می‌شوند. البته تعداد ورودی‌های هر بلوک منطقی متفاوت است و به نوع FPGA مربوط می‌شود. به عنوان مثال بلوک های منطقی در خانواده‌ی $ACT-1$، از نوع ۸ ورودی است. البته در برخی موارد به بلوک های منطقی، سلول های منطقی نیز گفته می‌شود. ساختار کلی FPGA به شکل ۱.۴ است.



شکل ۱.۴: نمایش ساختار درونی کلی FPGA

البته بسیاری از سلول های منطقی بر اساس جداول LUT ساخته می‌شوند. LUT از تعدادی سلولهای حافظه SRAM تشکیل می‌شود که در هنگام برنامه ریزی FPGA ، مقدار دهی می‌شوند. پیاده سازی توابع مختلف نیز به وسیله‌ی در کنار هم قرار گرفتن بلوک های منطقی و همچنین تنظیم ارتباط بین هر بلوک و به عهده گرفتن پردازش اطلاعات توسط هر بخش انجام می شود.

برای برنامه نویسی و طراحی FPGA ها از دو روش زبان های توصیف سخت افزار ($HDL-AHDL-VHDL$) و یا طراحی مدار(RTL) استفاده می شود. این تراشه ها می توانند چندصد میلیون گیت منطقی(قابل پیکربندی) داشته باشند که همین ویژگی آنها را برای پیاده سازی توابع پیچیده و بسیار بزرگ دلپذیرتر می کند. تعریفی دیگر که در FPGA به آن اشاره می‌شود، «آرایه گیت برنامه پذیر در محل» است، بعبارت دیگر میتوان گفت FPGA ها ISP هستند، یعنی سیستم‌های برنامه پذیر درون سیستمی! ISP

[۱]به قطعاتی گفته می شود که توان برنامه پذیری هنگام استقرار در سیستمی سطح بالاتر را داشته باشند. همین ویژگی، امکان تغییر طرح پیاده سازی شده را بصورت ساده برای ما فراهم می آورد بدون آنکه نیاز به تولید نسخه ی جدید باشد! در نتیجه زمان عرضه به بازار طرح بسیار کوتاه تر می شود. FPGA ها برای پیشرفت شرکت های startup (شرکت های نوپا و تازه تأسیس) بسیار مناسبند، چرا که حتی گروه های کوچک مهندسی در محیط های آزمایشگاهی کوچک مبتنی بر FPGA، موفق به اجرای طرح های خود می شوند. و همچنین هزینه های توسعه بسیار پایین تر از نمونه های مشابه است. در ابتدای ظهور FPGA ها (اواسط دهه ۱۹۹۰) از آنها برای پیاده سازی منطق اتصالی (*glue logic*) و ماشین هایی با پیچیدگی متوسط و پردازش داده های نسبتاً کم استفاده میشد. در اوایل دهه ۱۹۹۰ و با پیشرفت FPGA ها، از آنها برای ارتباط و شبکه، یعنی پردازش بلاک های بزرگ داده و فرستادن آنها به اطراف استفاده میشد؛ و در اواخر دهه ۱۹۹۰، بازار شاهد ورود آنها به کاربردهای صنعتی، لوازم خانگی و خودروسازی بود. اما امروزه از FPGA ها تقریبا برای پیاده سازی هرچیزی مانند دستگاه های مخابراتی، رادیوهای نرم افزاری، رادارها، پردازش تصویر و دیگر کاربردهای پردازش سیگنال(DSP) وحتی قطعات *System On Chip*حاوی عناصر نرم افزاری و سخت افزاری استفاده می شود.

۳.۴ تفاوت FPGA با سایر میکروپروسسورها

برای اینکه درک کنیم FPGA ها چه کاربردی داشته و چه جایگاهی دارند، در این قسمت به بیان تفاوت‌های بین FPGA ها و سایر میکروها خواهیم پرداخت تا بتوانیم کاربردهای مختلف این پروسسورها را بیشتر درک کنیم[۱۹] و [۲۰] .

در ابتدا از میکروپروسسورهای قدرتمند ARM شروع می‌کنیم: ARM یک میکروپروسسور ۳۲ بیتی با معماری RISC می‌باشد. $ARM7$ یک میکروپروسسور قوی با فرکانس کاری بین ۶۶ مگاهرتز تا ۱۳۳ مگا هرتز می‌باشد. در حالی که $ARM11$ یک میکروپروسسور با فرکانس کاری بین ۳۵۰ مگاهرتز تا ۵۵۰ مگاهرتز می‌باشد. این پروسسورها دارای کاربردهای مختلف و زیادی می‌باشند و بیشتر در سیستم‌های کاربر پسند که نیازمند ظاهری زیبا می‌باشند استفاده می‌شود. دلیل این امر این است که ARM برای اجرای سیستم عامل‌های مختلفی همچون UNIX و یا Windows بر روی IC بسیار مناسب هستند. کاربر می‌تواند برنامه‌های خود را در محیطی مانند برنامه‌ی $VisualStudio$ نوشته، به صورت فرمت اجرایی در قالب فرمت UNIX و یا Windows در آورده و سپس روی سیستم عامل این پردازنده آن را نصب کند. از کاربردهای فراوان این پروسسورها می‌توان به موارد زیر اشاره کرد:

[۱] in system programmable

۱ پایانه‌های بانکی POS

۲ سیستم‌های موقعیت یاب GPS

۳ پردازنده‌ی تلفن‌های همراه هوشمند همانند Apple ، Sony و ...

۴ دوربین‌های فیلم‌برداری و عکس‌برداری دیجیتال

۵ تبلت‌ها

۶ دستیار دیجیتالی شخصی PDA ها

در شکل زیر نمونه‌ای از وسایلی که در موارد فوق اشاره شد را ملاحظه می‌کنید. همانگونه که ملاحظه می‌شود بدون این پروسسورهای قدرتمند بسیاری از امکانات دیجیتالی که جهان پیرامون ما مملو از آنهاست و در راحتی زندگی به ما کمک می‌کنند و در اکثر لحظات زندگی همراه ما بوده و در اختیار ما هستند، با این قدرت و کیفیت کنونی، وجود نداشتند.

پروسسور قدرتمند بعدی که در اینجا کاربردهای آن را بررسی می‌کنیم، پروسسورهای DSP می‌باشند. واژه‌ی DSP اختصاری از ابتدای کلمات *Digital Signal Processor* می‌باشد. پردازش سیگنال های دیجیتال[۱] شامل نمایش دیجیتالی سیگنال‌ها و استفاده از سیستم‌های دیجیتالی برای تحلیل کردن، تغییر دادن، ذخیره سازی یا استخراج اطلاعات از سیگنال‌های دیجیتالی می‌باشد. DSP مدار مجتمعی است که برای پردازش اطلاعات و سیگنال‌ها با سرعت بالا طراحی شده است و در صدا، تصویر، مخابرات و ... کاربرد دارد.

پردازش سیگنال دیجیتالی به طور طبیعی نیازمند تعداد زیادی محاسبات به صورت $A = B \times C + D$ می‌باشد. یعنی محاسبات دیجیتالی اکثراً به صورت اعمال ضرب، جمع و سپس ذخیره‌سازی انجام می‌شود. شاید این عمل ساده به نظر برسد ولی وقتی نیاز به سرعت بالایی داشته باشیم و نیازمند آن باشیم تا این اعمال ساده در یک سیکل زمانی به اندازه‌ی کسری از میکروثانیه انجام گردند اختصاص دادن پروسسورهای قوی و سریعی همچون DSP ها به این کار، تحول بزرگی را در پردازش اطلاعات دیجیتالی بوجود آورده است. بیشتر DSP ها دستور عمل ویژه‌ای دارند که به آن‌ها اجازه می‌دهد که عملیات ضرب، جمع و ذخیره کردن نتیجه را در یک سیکل انجام دهند. این دستور عمل معمولاً MAC [۲] امروزه انواع مختلفی از پردازنده‌های DSP به صورت تجاری در دسترس هستند. در جدول ۱.۴ تعدادی از پردازنده‌های DSP به همراه نوع معماری آنها و همچنین فرکانس کاری خلاصه شده‌اند.

[۱] Digital Signal Processing
[۲] **M**ultiply,**AC**cumulate

جدول ۱.۴: برخی از DSP های موجود به همراه مشخصات مختصری از آنها

سازنده DSP	خانواده قطعه	معماری	سرعت کلاک
Texas Instrument	$TMS320C54x$	$Fixed-Point$	$160MHz$
	$TMS320C67x$	$Floating-Point$	$300MHz$
Analog Devices	$ADSP218x$	$Fixed-Point$	$80MHz$
	$ADSPTS20x$	$Fixed-FloatingPoint$	$600MHz$
$Freescale$	$MSC71x$	$Fixed-Point$	$200MHz$
	$MSC81x$	$Fixed-Point$	$400MHz$
$Agere$	$DSP1641x$	$Fixed-Point$	$285MHz$

این پردازنده‌های DSP شامل ضرب کننده‌های سخت افزاری و امکانات شیفت دادن هستند و در هر یک سیکل می توانند تا چندین میلیون دستورعمل را اجرا می‌کنند. برخی از DSP ها نیز امکان اجرای دستورات به صورت موازی را دارا می‌باشند و برخی هم توان مصرفی بسیار پایینی دارند. به دلیل این تنوع گسترده در این پردازنده‌ها، نوع کابرد آن ها در صنعت و بازار نیز گسترده و متنوع است که در زیر از خانواده‌ی TMS برخی از این کاربردها آورده شده‌اند.

- $TMS320-C2000$

ارزان قیمت بوده و در سیستم‌های کنترلی از قبیل کنترل موتور، کنترل دیجیتالی و... کاربرد دارند.

- $TMS320-C5000$

بازده‌ی بالایی داشته و در بین بقیه‌ی DSP ها دارای بیشترین MIPS[۱] می‌باشد و در تلفن‌های بیسیم، پخش کننده‌های بیسیم، دوربین‌های دیجیتال، مودم‌ها و... کاربرد دارند.

- $TMS320-C6000$

این گروه دارای عملکرد بالا و استفاده‌ی آسانی بوده و در زیر ساخت‌های مخابراتی، ایستگاه‌های بیسیم، پردازش تصویر و ویدئو و سرورهای چند رسانه‌ای کاربرد دارند.

پروسسور دیگری را که آن را معرفی کرده و آن را با پروسسورهای دیگر مقایسه می‌کنیم، FPGA ها هستند. FPGA ها با استفاده از معماری موازی بسیار قدرتمندی که دارند، قدرت محاسباتی چشمگیری را برای کارایی خیلی بالا فراهم می‌کنند. سخت افزار FPGA قابل پیکره‌بندی مجدد می‌باشد، بنابراین به طراحان سیستم این اجازه را می‌دهد که برای پیاده سازی الگوریتم‌هایی که نیازمند عملکرد بالا و قیمت تولید کم هستند، معماری سخت افزاری را بهینه‌سازی کنند. مزیت اصلی FPGA این است که دارای معماری بسیار انعطاف‌پذیری می‌باشند. اما این انعطاف‌پذیری با پرداخت هزینه‌ی بیشتر در مصرف قطعات داخلی سخت

[۱] Million Instruction Per Second

افزار همراه می‌باشد، یعنی انعطاف پذیری بیشتر برابر است با مصرف گیت‌های بیشتر، مساحت سیلیکون مصرفی بیشتر، منابع بیشتر برای مسیریابی و در نتیجه مصرف توان بیشتر. زبان مورد نیاز برای برنامه‌نویسی این پروسسورها زبان سخت افزار گرای VHDL و Verilog می‌باشد که نوشتن یک کد به این زبان نسبت به زبان C ۵ برابر بیشتر طول می‌کشد و سختی و دشواری خود را دارد. از جمله مزایای FPGA ها می‌توان به انعطاف‌پذیری بیشتر نسبت به ASIC ها و عملکرد بالا در کاربردهای خاص خود را اشاره کرد. همچنین FPGA ها این قابلیت را دارند که از سخت افزار یکسان برای کاربردهای مختلف بهره ببرند. اما این پروسسورهای قدرتمند نسبت به پروسسورهایی که معرفی کردیم، در مصرف توان مصرفی و همچنین قیمت آن ها در بازار، برتری ندارند. در زمینه‌ی تولید FPGA ۲ شرکت بزرگ Altera و Xilinx تقریباً تمامی بازار مربوط به این پروسسورها را در اختیار خود دارند. سری Spartan ساخت شرکت Xilinx شامل FPGA های ارزان قیمت و مناسب جهت کاربردهای پژوهشی برای دانشجویان و سری Virtex شامل FPGA های گران قیمت‌تر و با امکانات و عملکرد بیشتر جهت انجام پروژه‌های تجاری و حساس‌تر مناسب می‌باشند. از جمله مهمترین کاربردهای FPGA می‌توان به این مسئله اشاره کرد که در کارهایی که لازم است یک الگوریتم پردازشی روی تعداد زیادی داده به طور همزمان اعمال شود، استفاده از یک FPGA انتخاب مناسبی است.

در جدول زیر به برخی از این پروسسورها و امکانات داخلی آنها اشاره گردیده است.

جدول ۲.۴: مشخصات برخی FPGA های خانواده‌ی Xilinx

	Spartan-۳	spartan-۶	Artix-۷	Kintex-۷	Virtex-۷	Kintex-Ultra	Virtex-Ultra
Logic Cells	$17,344$	$147,443$	$215,360$	$477,760$	$1,954,560$	$1,160,880$	$4,407,480$
DSP Slice	28	180	740	$1,920$	$3,600$	$5,520$	$2,880$
Block RAM	$1.1Mb$	$4.8Mb$	$13Mb$	$34Mb$	$68Mb$	$76Mb$	$115Mb$
I/O Pins	190	576	500	500	$1,200$	832	$1,456$
I/O Voltage	$1.2-3.3V$	$1.2-3.3V$	$1.2-3.3V$	$1.2-3.3V$	$1.2-3.3V$	$1.0-3.3V$	$1.0-3.3V$

به طور کلی:

۱ درکاربردهایی که نیاز است یک الگوریتم پردازشی روی تعداد زیادی از داده به طور همزمان اعمال شود، FPGA ها مناسب می‌باشند.

۲ در کاربردهایی که نیازمند سرعت بالا، حجم پردازشی زیاد و پیچیده می‌باشیم، مانند کارهای پردازش سیگنال (صدا، تصویر و...) پردازنده‌های DSP مناسب می‌باشند.

۳ برای کاربردهایی که ترکیبی از موارد بالا هستند، ترکیب پردازنده‌ها گزینه‌ی مناسبی می‌باشد، برای مثال در کاربردی که نیاز به حجم پردازشی بالا و محیط کاربرپسند دارند، پردازنده‌های OMAP ساخت شرکت TI که ترکیبی از پردازنده‌های DSP و ARM می‌باشد، مناسب هستند.

فصل ۵
الگوریتم پیشنهادی

۱.۵ گزینش داده‌ها به کمک Adaboost

یکی از کارهای مقدماتی که انجام آن لازمه‌ی انجام هر عملیات پردازش روی تصاویر می‌باشد، استخراج ویژگی از تصاویر موجود و سپس اجرای کدها و الگوریتم‌های مدنظر بر روی آنها می‌باشد. لذا در این پایان نامه هم مروری اجمالی برروی مجموعه داده‌های مورد استفاده، استخراج ویژگی، عملیات انجام شده برروی این ویژگی‌ها و گزینش آنها و سایر کارهای موردنظر، خواهیم کرد.
روش‌های متفاوت زیادی برای انجام استخراج ویژگی استفاده می‌شود که در زیر به چند نمونه از آنها اشاره می‌شود:

۱ $ForwardFeatureSelection$

۲ $BackwardFeatureSelection$

۳ $FloatingPointFeatureSelection(Backward\ \&\ Forward)$

۴ $FDR(FeatureDiscriminantRatio)$

۵ $GA(GeneticAlgorithm)$

۶ $AdaBoost$

در این پایان نامه از مجموعه داده‌های معتبر FERET [۲۷] جهت آموزش شبکه‌ی عصبی بر روی آنها استفاده خواهیم کرد و سپس چند تصویر آزمایش جهت انجام آزمایش تابع هدف بهره خواهیم برد. در این پایان نامه درباره‌ی روش خاصی در مورد شناسایی چهره در تصاویر خواهیم پرداخت که نتایج بسیار موفقی

را در این زمینه کسب کرده است. این روش اولین بار در مقاله‌ای در سال ۲۰۰۳ میلادی توسط ۲ شخص به نام‌های ویولا و جونز[۱] ارائه شد، که در طی سال‌های اخیر بیش از ۳۰۰ بار در مقالات مختلف ذکر شده و یا استفاده شده است.

ما نیز در این پایان نامه در مرحله‌ی پیش پردازش از این روش استفاده خواهیم کرد و ویژگی‌های استخراج شده به کمک فیلترهای هار را به کمک این روش گزینش می‌کنیم و سپس در الگوریتم خود از نتایج این روش استفاده کرده و بقیه‌ی روش خود را برای شناسایی چهره در تصاویر به کمک ماشین بردار پشتیبان موازیِ خود پی می‌گیریم.

در این قسمت توضیح خواهیم داد که به چه دلیل الگوریتم AdaBoost و همچنین ویژگی‌های ساده‌ی هار را به عنوان مبنا و مقدمه‌ی کار خود برگزیدیم. از آنجا که قصد داریم تصاویر را با هر پس زمینه‌ای و همچنین با کیفیت‌های اغلب پایین (مثلاً تصاویر با اندازه‌ی ۲۴×۲۴)، این طور به نظر می‌رسد که می‌توان به کمک الگوریتم‌های هندسی بسیار بهتر به جوابِ مناسب رسید. اما ما در اینجا علاقه‌ای به روش‌های هندسی نداریم زیرا ما در مجموعه داده‌های خود اغلب تصاویر چهره از زاویه‌ی روبروی چهره‌ی شخص را داریم و به همین دلیل که زاویه‌ی شخص در مجموعه داده‌ی ما تغییر بسیار نامحسوسی دارد، می‌توان به کمک الگوریتم‌های آموزشی در شبکه‌های عصبی همانند این الگوریتم و همچنین به دلیل اینکه این الگوریتم درصد خطا را تا میزان صفر کاهش می‌دهد، این روش هم به نظر روشی مطمئن و قابل اعتمادی به نظر می‌رسد. این الگوریتم از نوع الگوریتم‌های تکرارشونده[۲] است که در هر بار تکرار خود، ترکیبی از کلاسه‌بندهای ساده[۳] را انتخاب می کند و در نهایت یک کلاسه‌بند بسیار قوی[۴] می‌سازد و به درصد موفقیت بسیار بالایی دست پیدا می‌کند. لذا ما از لحاظ تئوری کاملاً مطمئن هستیم که روشی را که برای گزینش ویژگی‌های خود برگزیدیم، کاملاً موفقیت آمیز بوده و در این مرحله درصد خطای بسیار کم و بازده‌ی بسیار بالا را در مقابل طولانی‌تر شدن زمان این الگوریتم در مقابل بقیه‌ی الگوریتم‌ها، خواهیم داشت.

همانطور که می‌دانید، روش‌های متفاوت زیادی جهت تهیه و استخراج ویژگی از تصاویر وجود دارد و استفاده از هرکدام از این ویژگی‌ها مزایا و معایب خاص خود را دارند. یکی از ساده‌ترین و مبتدی‌ترین روش‌های استخراج ویژگی، استفاده از پیکسل‌های خام هر تصویر می‌باشد. یعنی به بیان دیگر هر ویژگی، تک تک پیکسل های هر تصویر می‌باشد.

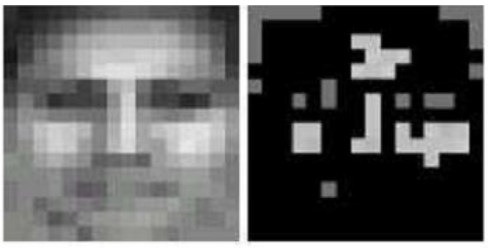
این روش اولین بار در شناسایی چهره توسط شخص پاولویک[۵] انجام شد. او در مقاله‌ی خود[۲۸] همانندِ کارِ پیش روی ما، بعد از استفاده از پیکسل‌های خام تصویر، از الگوریتم AdaBoost جهت آموزش ویژگی‌ها

[۱] Paul Viola and Micheal Jones
[۲] iterative
[۳] Weak Classifier
[۴] Strong Classifier
[۵] Pavlovic

استفاده نمود. او نحوه‌ی تعریف کلاسه‌بندهای ضعیف خود را بصورت رابطه‌ی زیر تعریف کرد:

$$h_k \in \{t(X\,|\theta, l) = sign(X^{(l)} - \theta)\}$$

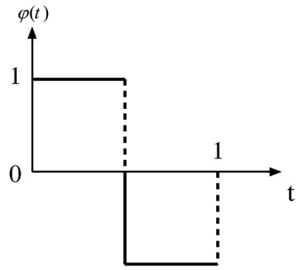
که در آن X به مجموعه بردارهای پیکسل‌های سیاه و سفید خام تصاویر اشاره می‌کند و $X^{(l)}$ نشان دهنده‌ی l اُمین پیکسل از مجموعه پیکسل‌هاست. کلاسه‌بند موردنظر در این روش مجموعه پیکسل‌های هر تصویر را با توجه به آنچه که توسط الگوریتم آموزشAdaBoost دیده است، تصمیم‌گیری می‌کند که پیکسل‌های وارد شده به چهره نزدیکتر می‌باشند یا به غیرچهره. همانطور که در شکل زیر هم دیده می‌شود، کلاسه‌بند با در نظر گرفتن پیکسل‌های وارد شده و با توجه به چیزی که فراگرفته است شکل ۱.۵ را در دسته‌ی چهره کلاسه‌بندی می‌کند.



شکل ۱.۵: نحوه‌ی گزینش بهترین پیکسل‌ها توسط الگوریتم AdaBoost از روی پیکسل‌های خام ورودی

یکی از مشکلات این روش این است که به عنوان مثال اگر یک تصویر با ابعاد ۲۰×۲۰ داشته باشیم، یعنی ۴۰۰ پیکسل در اختیار داریم که می بایست برای هر پیکسل یک کلاسه‌بند ضعیف ایجاد کنیم تا در نهایت به تعداد محدودتری کلاسه‌بند قوی دست یابیم که همین امر باعث می‌شود تا به طور چشمگیری مدت زمان آموزش افزایش پیدا کند.

یکی دیگر از روش‌های ساده ولی بسیار مفید و موفق، استفاده از فیلترهای هار می‌باشد. در ادامه مروری بر نحوه‌ی استخراج ویژگی به کمک این فیلترها خواهیم پرداخت.



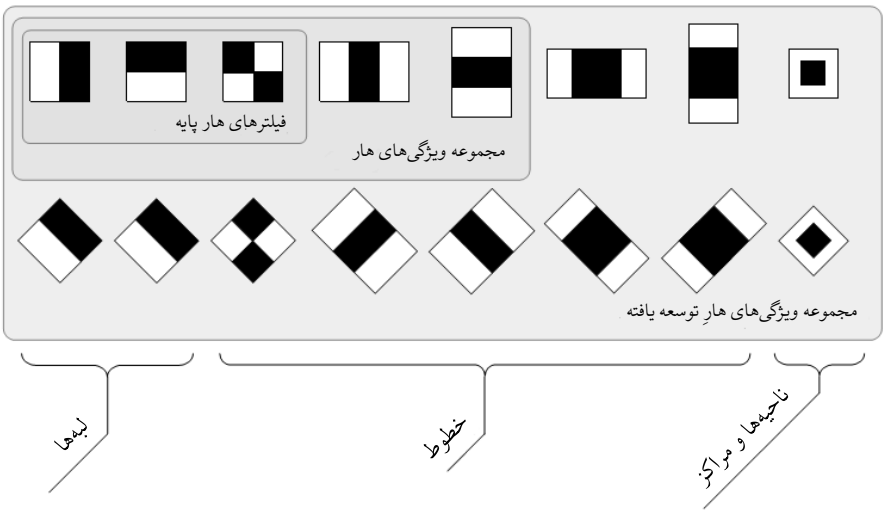
شکل ۲.۵: فیلتر هار طبق تعریف رابطه‌ی ۱.۶

در این پایان نامه که هدف، پیاده‌سازی سخت‌افزاری است و با توجه به اینکه پیکسل‌های خام به عنوان ویژگی یک روش زمان‌بر و مقدماتی است، لذا در اینجا از روش استخراج ویژگی دیگری استفاده خواهیم کرد که هم ساده بوده و هم در زمان آموزش صرفه‌جویی گردد. لذا در میان روش‌های مختلف استخراج ویژگی روشی ساده ولی دقیق و کارا را استفاده خواهیم کرد که آن، روش استخراج ویژگی به کمک فیلترهای هار می‌باشد. اساس و پایه‌ی مفهوم فیلترهای هار همان است که در مباحث موجک[۱] و بحث موجک هار به عنوان یکی از ساده‌ترین و مقدماتی‌ترین موجک‌ها به آن اشاره می‌شود که به دلیل پرهیز از شلوغیِ بحث از ذکر جزئیات صرفنظر می‌شود. در سیستم شناسایی چهره، این ویژگی‌هایِ بسیار ساده، بسیار به کار خواهند آمد. اگر از موجک‌ها یادآوری کوچکی داشته باشیم، موجک هار را می‌توان به صورت زیر تعریف کرد.

$$\varphi(x) = \begin{cases} 1 & 0 \leq x < \frac{1}{2} \\ -1 & \frac{1}{2} \leq x < 1 \\ 0 & otherwise \end{cases} \qquad (۱.۵)$$

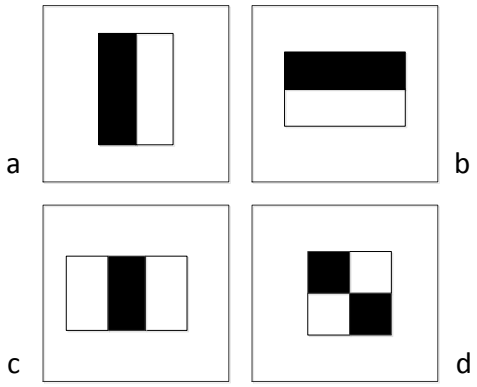
در شکل زیر فیلتر هار که در بالا تعریف شد را ملاحظه می‌کنید: در پردازش تصویر، این شکل موج را به صورت مستطیلی به صورت سفید و سیاه نشان می‌دهیم، که قسمت تیره را با ضریب منفی و قسمت روشن را با ضریب مثبت در نظر گرفته و بر پیکسل‌های تصویر اعمال می‌کنیم. در شکل زیر انواع مختلفی از فیلترهای هار را ملاحظه می‌کنید. این فیلترها اغلبِ ویژگی‌هایِ تصویر را استخراج می‌کنند و در برخی مقالات معتبر از آنها جهت شناسایی تصویر استفاده شده است.

[۱] Wavelet



شکل ۳.۵: فیلترهای هار مستطیل شکل

در میان تعداد زیاد فیلترهای هار ما در این پایان نامه بر اساس مرجع معتبر [۲۹] که تنها از فیلترهای مشخص شده در شکل زیر استفاده کرده است، استفاده می‌کنیم. به این ترتیب به کمک فیلترهای هار، بسیاری از ویژگی‌های ساده از یک تصویر استخراج می‌شود که سرانجام به کمک الگوریتم AdaBoost بهترین ویژگی‌های تصویر و غیر تصویر از میان انبوه ویژگی‌ها استخراج شده و به عنوان مناسب‌ترین ویژگی‌ها به کلاسه‌بند ماشین بردار پشتیبان اعمال می‌کنیم تا در نهایت به کمک آن بتوانیم کلاسه‌بندی خوب و موفقی از تصاویر چهره و غیرچهره داشته باشیم. چگونگی محاسبات مربوط به استخراج ویژگی به کمک مستطیل‌های فوق را در [۳۰] می‌توانید دنبال کنید.



شکل ۴.۵: فیلترهای هار مستطیل شکل

۲.۵ مقدمه‌ای بر تئوری ماشین بردار پشتیبان موازی (PSVM)

در این قسمت الگوریتم ماشین بردار پشتیبان موازی را که هدف این پایان نامه می‌باشد را تشریح خواهیم کرد. در الگوریتمی که بیان خواهیم نمود ماشین بردار پشتیبان (SVM) را به طور مؤثری به صورت موازی در خواهیم آورد که ضمن دارا بودن دقت ماشین بردار پشتیبان معمولی از لحاظ سرعت انجام محاسبات عملیات مرحله‌ی آزمایش را بطور محسوسی بهبود خواهد بخشید. از مزایای دیگر الگوریتم پیشنهادی باید خاطر نشان کرد که در این الگوریتم به دلیل تقسیم کردن داده‌ها به چند قسمت و انجام محاسبات برای مسائلی که دارای داده‌های بسیار زیادی نسبت به حالت عادی می‌باشند بسیار مفید خواهد بود و می‌تواند تعداد بسیار بیشتری از داده‌ها را به عنوان ورودی بپذیرد در حالیکه ماشین بردار پشتیبان معمولی کل مجموعه‌ی بردارهای ورودی را بصورت یکجا و کلی مورد پردازش قرار می‌دهد که برای داده‌های با تعداد بالا مشکل‌ساز خواهد بود.
همانطور که می‌دانیم یکی از خصوصیات ماشین بردار پشتیبان، تئوری ریاضیاتی قوی این کلاسه‌بند می‌باشد و اگر به فصل ۲ این پایان نامه دوباره نگاهی بیاندازیم درخواهیم یافت که محاسبات لازم برای اجرای مرحله‌ی آزمایش نیز دارای پیچیدگی‌های ریاضیاتی می‌باشد. لذا یکی از راهکارها برای اجرای موازی‌سازی بر روی ساختار این کلاسه‌بند، وارد شدن به ساختار تئوری و ریاضیاتی آن می‌باشد. خارج کردن محاسبات ریاضی از حالت عادی و اجرای آنها بصورت پردازش موازی می‌تواند یکی از راهکارهای مطمئن برای پیاده‌سازی PSVM باشد. برای مثال الگوریتمی را که در مرجع [۳۱] ذکر شده است را مرور خواهیم کرد. مروری بر تلاش‌های دانشمندان و مهندسین در این زمینه می‌تواند هم در یافتن راه‌حل بهینه و هم در تسلط پیدا کردن بر الگوریتم پیشنهادی خود کمک شایانی خواهد نمود.
نکته‌ای که در اینجا از بحث تئوری ماشین بردار پشتیبان موازی (PSVM) باید ذکر شود این است که ما در این پایان نامه به دلیل پیچیدگی محاسبات نیاز داریم تا پارامترها و پیچیدگی‌های مسئله را بتوانیم به حداقل تعداد ممکن کاهش دهیم. یکی از راهکارها برای این منظور حذف کردن پارامتر C از شروط مرزی حل معادله‌ی درجه ۲ می‌باشد. برای جلوگیری از مباحث پیچیده و ریاضیاتی تنها این را ذکر خواهیم کرد که با انجام نرمالیزاسیون زیر و اعمال آن به معادله‌ی درجه ۲ که به عنوان تابع هدف آن را در فصل ۲ بدست آوردیم، می‌توانیم پارامتر C را کنار بگذاریم.

$$\beta_i = \alpha_i \frac{2^n - 1}{C}$$

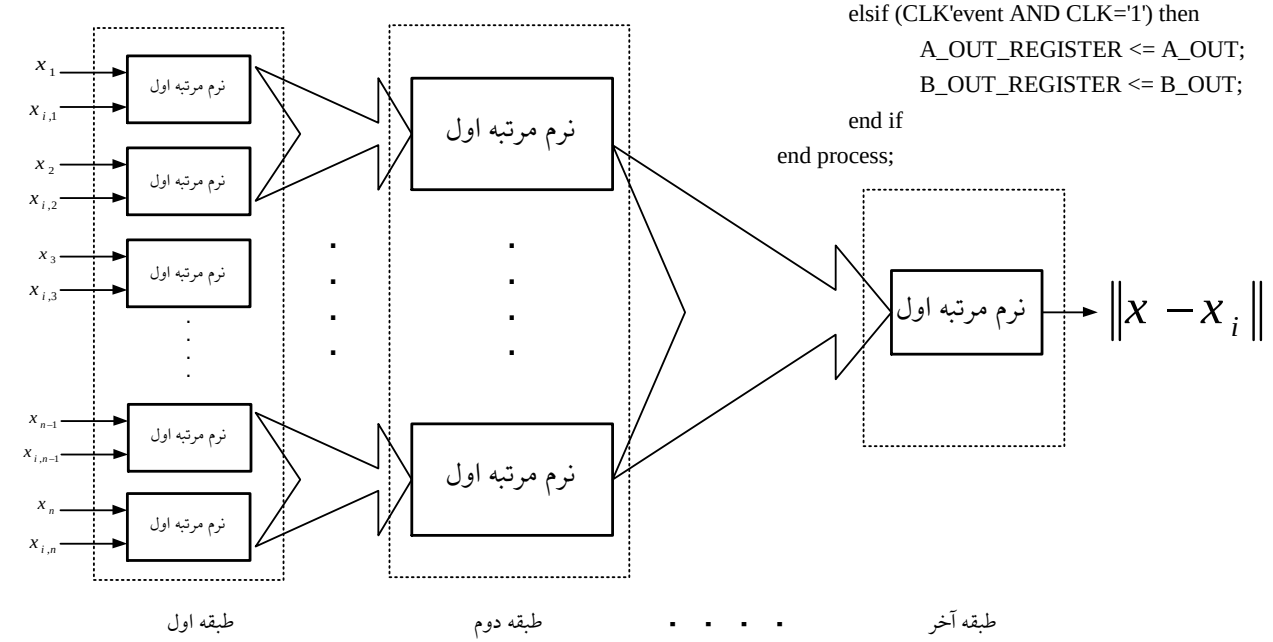
که در آن مقدار n مقدار بیتی است که برای محاسبه‌ی α_i ها تعیین نمودیم. برای اطلاعات بیشتر می‌توانید مباحث و اثبات ریاضی این نرمالیزه کردن را در مرجع ذکر شده مطالعه نمایید. در رابطه با مرحله‌ی آموزش ماشین بردار پشتیبان و بهینه‌سازی آن مقالات متعدد و معتبری وجود دارند منتها قصد ما در اینجا وارد شدن به تمامی ابعاد حل مسئله‌ی کلاسه‌بند SVM نمی‌باشد و تنها وارد مباحث مربوط به بخش آزمایش

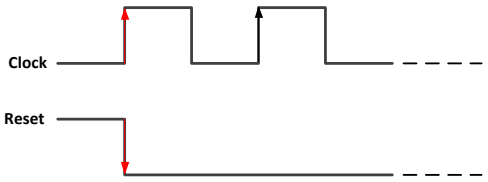
ماشین بردار پشتیبان خواهیم شد و تنها سعی خواهیم کرد این مرحله را مورد تجزیه و تحلیل قرار داده و با موازی‌سازی عملکرد آن را بهبود بخشیم. فرض را بر آن می‌گیریم که مرحله‌ی آموزش به کمک نرم‌افزار انجام شده و نتایج و پارامترهای آن در دسترس ما می‌باشند. از جمله پارامترهایی که مقدار آن برای ما حائز اهمیت است مقادیر x_i بردارهای پشتیبان، ضرایب α_i و برچسب کلاس‌ها y_i می‌باشد. همانگونه که از فصل تئوری ماشین بردار پشتیبان به خاطر داریم بردارهای آزمایش به عنوان ورودی وارد شده و مقدار فاصله‌ی آنها از تک تک بردارهای پشتیبان بدست آمده (نرم مرتبه اول) سپس در ضرایب آلفا و برچسب کلاس‌ها ضرب شده و با یکدیگر جمع می‌گردند.

اگر بتوان به نحوی روابط ریاضی حاکم بر تئوری ماشین بردار پشتیبان را بصورت موازی در بیاوریم، می توان الگوریتم PSVM را توسعه و گسترش داد. طبق تحقیقات انجام شده در مقاله‌های پیشین از جمله مراجع [۳۲] و [۳۳] یکی از روابط ریاضی را که می‌توان تئوری ریاضیاتی آن را بصورت موازی درآورد، تابع نرم مرتبه اول می‌باشد. تابع نرم مرتبه اول در دل تابع کرنل قرار می‌گیرد. اگر بلوک محاسباتی کرنل (گاوسی، سخت‌افزار پسند) را به چندین بلوک محاسباتی تقسیم کنیم مهمترین بلوک آن، بلوک نرم مرتبه اول می‌باشد. بنابراین در الگوریتم PSVM یکی از بلوک‌های محاسباتی که زمان زیادی را از پروسسور می‌گیرد و گلوگاه محاسبات پیچیده می‌باشد، همین بلوک نرم مرتبه اول می‌باشد. بنابراین روند این نوشته بر آن منوال شد تا بتوانیم این بلوک را بصورت موازی در دل بلوک کرنل در الگوریتم PSVM پیاده‌سازی نماییم. در شکل زیر نمای کلی از این هدف را مشاهده می‌نمایید و در قسمت بعدی به طور مفصل به این مطلب خواهیم پرداخت و کد نوشته شده برای پیاده‌سازی هسته‌ی اصلی الگوریتم خود را شرح خواهیم داد.

اگر به شکل صفحه‌ی بعد نگاهی بیاندازید، الگوریتم کلی برای اجرای ماشین بردار پشتیبان به صورت موازی که مدنظر داریم را ملاحظه خواهید کرد. بدین ترتیب که با وارد شدن به داخل بلوک محاسباتی کرنل و اجرای تاب نرم مرتبه اول در چند طبقه و تعریف هر کدام در Process های متفاوت می‌توان آنها را به صورت موازی در آورد. از مهمترین خواص FPGA ها نیز همین است که Process ها را به صورت موازی اجرا می‌کند و ما نیز از همین جنبه استفاده خواهیم نمود که در فصل بعدی به طور مفصل‌تری توضیح داده خواهد شد.

```
process(CLK, RESET)
    begin
        if (RESET = '1') then
            A_OUT_REGISTER <= (others => '0');
            B_OUT_REGISTER <= (others => '0');
        elsif (CLK'event AND CLK='1') then
            A_OUT_REGISTER <= A_OUT;
            B_OUT_REGISTER <= B_OUT;
        end if
end process;
```






```
if (CLK'event AND CLK='1') then
        for i in 1 to 4 loop
            ABSOLUTE_DIFFERENCE_REGISTER(i) <= ABSOLUTE_DIFFERENCE_VALUE(i);
        end loop;
```

شکل ۵.۵: نمای کلی از موازی‌سازی کرنل مربوط به الگوریتم PSVM

۳.۵ پیاده‌سازی ماشین بردار پشتیبان موازی بر روی سخت‌افزار

۱.۳.۵ دلیل انتخاب FPGA برای پیاده‌سازی

هدف ساخت سیستم‌های تعبیه شده بر روی سخت‌افزار[۱] برای کابردهای آنی و بلادرنگ باعث شده است تا دانش و اطلاعات در زمینه‌ی ساخت و پیاده‌سازی الگوریتم‌های هوش مصنوعی بر روی چیپ‌ها و IC ها فزونی یابد. برای مثال، الگوریتم‌های شناسایی الگو و یا الگوریتم‌های یادگیری ماشین از جمله این دانش‌ها می‌باشند. این الگوریتم‌ها نیازمند قدرت محاسیاتی بسیار سنگین و پیچیده‌ای می‌باشند که همانطور که می‌دانیم این قدرت محاسباتی هم اکنون بر روی بردهای سخت‌افزاری موجود نمی‌باشند و با کمبود منابع سخت‌افزاری و محدودیت آنها مواجه هستیم، به عنوان مثال اگر بخواهیم برد سخت‌افزاری را دقیقاً متناسب با قدرت محاسباتی که نیاز داریم بسازیم هم از لحاظ توان مصرفی و هم از لحاظ اندازه و سایز چیپ به مشکلات اساسی بر می‌خوریم.

یکی از راهکارها جهت کاهش اثرات این مشکل این است که پیچیدگی‌های موجود در محاسبات مربوط به الگوریتم‌ها را طوری کاهش دهیم که جهت پیاده‌سازی روی سخت‌افزار مناسب گردند. یکی دیگر از راه‌های غلبه بر این مشکل این است که منابع سخت‌افزاری در دسترس را طوری مهیا کنیم و طوری آنها را پیکره‌بندی کنیم که کاملاً با هدف محاسبات ما همخوانی داشته باشد. به عنوان مثال، واحد پردازش اطلاعات (CPU) در حالی که با یک واحد پردازشگر اطلاعات جانبی به صورت همکار به واحد پردازش اطلاعات ما کمک می‌کند در اینصورت بازده‌ی نهایی پیاده‌سازی الگوریتم پیچیده بسیار بیشتر از حالتی می‌شود که یک واحد پردازشگر اطلاعات همه منظوره وظیفه‌ی کلی انجام این محاسبات را بر عهده بگیرد.

یکی از بهترین انتخاب‌ها برای این منظور، استفاده از FPGA هاست. این پروسسورها می‌توانند در راستای پیاده‌سازی محاسبات پیچیده‌ی ما، پیکره‌بندی گردند. اگر از واحد پردازش FPGA در کنار پروسسور اصلی سخت‌افزار خود استفاده کنیم می‌توان محاسبات بسیار پیچیده با حجم بالای پردازشی را به FPGA سپرد، در اینصورت پروسسور اصلی می‌تواند بسیاری از کارهای دیگر را که به آن محوّل می‌شود (به عنوان مثال رابط‌های گرافیکی و یا ارتباطات شبکه، ورودی خروجی‌ها و ...) را به خوبی انجام دهد و دیگر درگیر کارهای محاسباتی پیچیده و و پردازش‌های سنگین نشود. در بین الگوریتم‌هایی که برای پیاده‌سازی شناسایی چهره بر روی سخت‌افزار مناسب می‌باشند، در اینجا از ماشین بردار پشتیبان استفاده خواهیم کرد. ماشین بردار پشتیبان به طور کلی و در اساس جهت انجام الگوریتم‌های شناسایی الگو و دیگر الگوریتم‌های هوش مصنوعی ساخته و توسعه داده شده است. در واقع ماشین بردار پشتیبان با شبکه‌های عصبی RBF و MLP که از معروفترین شبکه‌های عصبی می‌باشند، شباهت بسیار زیادی دارد. ماشین بردار پشتیبان بر اساس نظریه‌ی

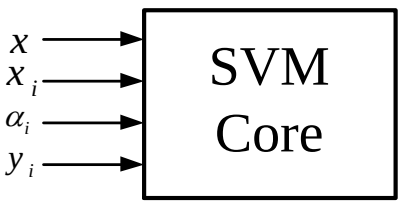
[۱] Embedded System

احتمالی و آماری در سال‌های دهه‌ی ۹۰ میلادی ایجاد و توسعه داده شد. این الگوریتم در مرحله‌ی آموزش خود، یک کلاسه‌بندی قوی را تشکیل می‌دهد که کلاس‌ها را بر اساس برچسب‌هایی که به آن کلاس‌ها چسبانده شده است، فرا می‌گیرد. سپس کلاسه‌بند آماده‌ی آزمایش و نتیجه‌گیری نهایی از داده‌های آزمایش می‌باشد تا مشاهده شود تا چه اندازه کلاسه‌بند داده‌های خود را درست کلاسه‌بندی کرده و تا چه اندازه به داده‌های آزمایش پاسخ درست می‌دهد که ما این مرحله، یعنی مرحله‌ی آزمایش را به عهده‌ی سخت‌افزار می‌سپاریم.

متأسفانه پیاده‌سازی ماشین بردار پشتیبان در مرحله‌ی آزمایش، بر روی سخت افزار کار ساده‌ای نمی‌باشد و نیازمند یکسری اقدامات اولیه و حل کردن برخی مشکلات است که شخص با آن روبرو می‌شود همانند کم بودن منابع سخت‌افزاری، در نظر گرفتن توان مصرفی مدار و دیگر مشکلات پیاده‌سازی همانند گیت‌ها، حافظه مورد دسترس، جمع‌کننده‌ها، ضرب‌کننده‌های داخلی و ... اشاره کرد، بنابراین نیاز نیست تا برای پیاده‌سازی بر روی سخت افزار بتوانیم بهترین و بهینه‌ترین کد سخت‌افزار گرا را برای سخت‌افزار نوشت تا بتوانیم منابع محدودی که در دسترس داریم را کنترل کنیم.

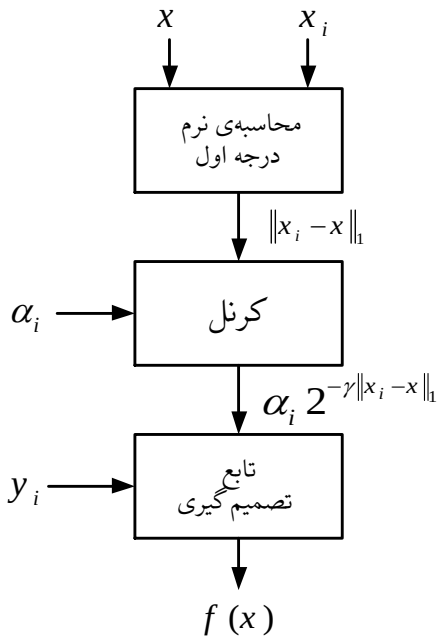
۲.۳.۵ معماری بلوک‌های طراحی شده در PSVM

در شکل ۶.۵ نمایی کلی از یک هسته‌ی SVM ارائه شده است. داده‌هایی که نیاز است تا در هسته‌ی SVM پردازش شوند همانند ضرائب α_i و بردارهای آموزش یافته (بردارهای پشتیبانx_i) می‌توانند در حافظه‌ی داخلی خودِ FPGA و یا در یک RAM خارجی نگاه داشته شوند. در هر ۲ حالت، معماری محاسبه‌ی مرحله‌ی آزمایش PSVM به کمک این داده‌ها یکسان است و تفاوت نمی‌کند از چه حافظه‌ای برای ذخیره‌ی داده‌های خود استفاده کنیم و این بستگی به نوع FPGA مورد استفاده‌ی ما دارد و مقدار حافظه‌ی داخلی Flash آن دارد. هسته‌ی پردازش SVM از ۳ قسمت تشکیل می‌گردد که در شکل ۶.۵ نشان داده شده است.



شکل ۶.۵: نمای کلی یک هسته‌ی SVM

که در آن اولین بلوک مقدار نرم $\| x - x_i \|_1$ را بین الگوهای ورودی x و i اُمین بردار پشتیبان x_i که در مرحله‌ی آموزش بدست آمده‌اند را محاسبه می‌کند. در مرحله‌ی بعد مقدار پارامتر ثابت γ را تعیین می‌کنیم که معمولاً مقدار آن را 0.5 در نظر می‌گیریم.

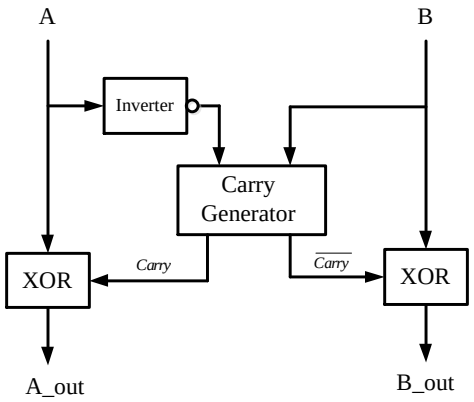


شکل ۷.۵: اجزای تشکیل دهنده‌ی هسته‌ی SVM

دومین بلوک مقدار تابع کرنل را محاسبه می‌کند و سومین بلوک هم تابع تصمیم‌گیری نهایی می‌باشد که تعیین می‌کند الگوهای وارد شده به عنوان ورودی جزء کلاس ۱ دسته‌بندی خواهند شد یا کلاس ۱-.
در ادامه به بررسی و تشریح نحوه‌ی عملکرد هرکدام از بلوک‌ها خواهیم پرداخت.

۱ بلوک Manhattan : این بلوک آنچه را که به عنوان نرم درجه اول و یا تفاوت مطلق مجموع[۱] می‌شناسیم را انجام می‌دهد، بین الگوهای ورودی و i اُمین بردار پشتیبان ورودی x_i . اولین مرحله از SAD ، مقدار اختلاف مطلق بین ۲ مقدار را محاسبه می‌کند که برای این کار از روش ارائه شده در مرجع [۴۰] استفاده شده است. برای محاسبه‌ی مقدار $| A - B |$ ضروری است تا مقادیر ۲ پارامتر A_{out} و B_{out} همانطور که در شکل زیر نشان داده شده است با یکدیگر جمع گردند. از آنجاییکه مرحله‌ی سخت و دشوار پیاده‌سازی مرحله‌ی آزمایش بردار پشتیبان در واقع همین مرحله‌ی بدست آوردن نرم درجه اول می‌باشد، بنابراین باید بیشترین بهینه‌سازی و ساده‌سازی در این مرحله صورت گیرد. همچنین اساس کار این پایان نامه،یعنی موازی سازی را نیز می‌توانیم در این مرحله انجام دهیم.

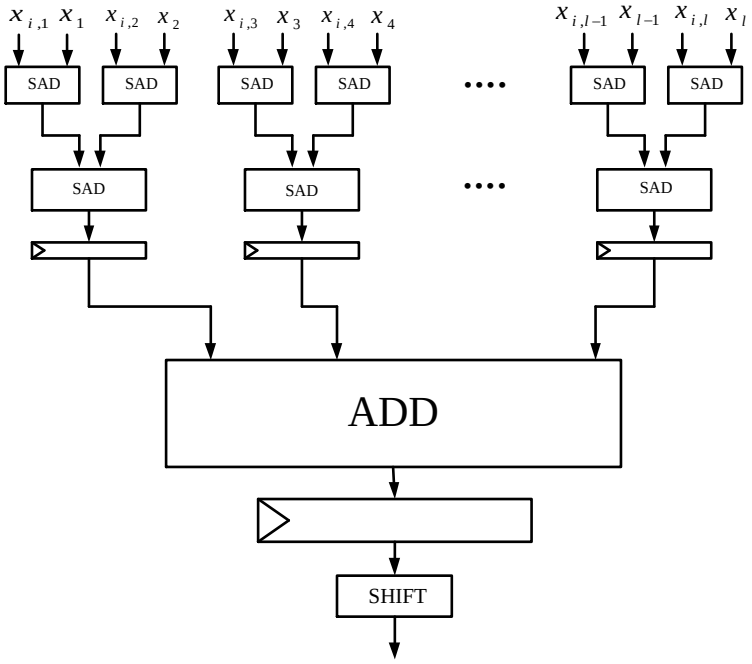
[۱] Sum of Absolute Difference



شکل ۸.۵: معماری داخلی SAD

در این مرحله می‌توانیم با ارائه‌ی الگوریتم ماشین بردار پشتیبان موازی بهترین عملکرد زمانی را داشته باشیم. همانطور که در شکل زیر آورده شده است بدست آوردن نرم پارامترهای x و x_i بوسیله‌ی تعدادی جمع کننده که در نهایت مقدار محاسبه شده‌ی نرم‌ها را با هم به طور موازی جمع کرده و به عنوان نتیجه‌ی نرم گزارش می‌دهد. این مرحله یکی از مراحلی است که در آن موازی سازی صورت می‌گیرد.

در این مرحله، بردارهای پشتیبان i اُم ، با نمونه‌های وارد شده از نمونه‌ی آزمایش مورد بررسی قرار گرفته و نتیجه‌ی حاصل به صورت موازی به کمک جمع کننده‌هایی با یکدیگر جمع می‌شود و در انباره‌هایی ذخیره می‌گردد، در نهایت مقدار تمامی این انباره‌ها، توسط جمع کننده‌ی نهایی با یکدیگر جمع شده و در نهایت نیز با عملگر شیفت مقدار γ را ایجاد می‌کنیم زیرا می‌توان مقدار گاما را طوری تعیین کرد به به جای اینکه مجبور باشیم از ضرب کننده برای ضرب یک عدد ثابت استفاده کنیم می توان عدد تقریبی از توان ۲ را که نزدیک به عدد 0.5 می باشد را ایجاد کنیم و از آن و به کمک یک عملگر شیفت که بسیار کم هزینه‌تر از عملگر ضرب می‌باشد، استفاده نماییم.



شکل ۹.۵: محاسبات مربوط به محاسبه‌ی نرم مرتبه اول به روش موازی

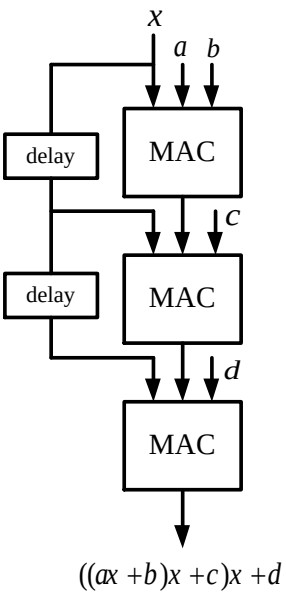
۲ بلوک محاسباتی کرنل: در حالت کلی، در این قسمت می‌خواهیم بلوکی را بسازیم که مقدار x^y را محاسبه می‌کند. در این پایان نامه می‌توانیم بجای x از مقدار ۲ و بجای y هم از مقدار اختلاف نرم استفاده کنیم. در محاسبات مربوط به کرنل در جواب نهایی تفاوت زیادی در این مورد که پایه‌ی توان عدد نپر e باشد با اینکه پایه‌ی توان عدد ۲ باشد، وجود نخواهد داشت. اگر مقدار خروجی مرحله‌ی قبل را E بنامیم، می‌توان محاسبات را به صورت زیر تشریح کرد:

$$2^{-\gamma\|x_i-x\|_1} = 2^E$$

مزیت این روش بر حالت استفاده از کرنل گاوسی، یعنی استفاده از پایه‌ی توان e این است که می‌توان پایه‌ی ۲ را با یک عمل شیفت که بسیار ارزان تمام می‌شود انجام داد در حالیکه برای پیاده‌سازی پایه‌ی نمایی مشکلات بسیار زیادی داریم و هزینه‌ی زیادی را می‌بایست متحمل شویم.

در ادامه،۳ نوع معماری پیاده‌سازی کرنل را معرفی خواهیم کرد که بستگی به نحوه‌ی استفاده‌ی کاربر و بسته به کاربرد ماشین بردار پشتیبان و نوع FPGA ای که کاربر می‌خواهد استفاده کند دارد. موازنه بین سرعت و میزان منابع مصرفی در انتخاب این معماری‌های متفاوت بسیار تأثیرگذار هستند. در زیر این ۳ نوع معماری معرفی خواهند شد:

الف **تخمین چندجمله‌ای:** در این معماری، تابع کرنل به کمک تخمین چندجمله‌ای محاسبه خواهد شد. به این ترتیب، نتیجه‌ی مقدار تابع کرنل با طبقاتی از واحدهای محاسباتی MAC [۱] محاسبه می‌شود. شکل زیر نحوه‌ی محاسبه‌ی تابع کرنل را به کمک این معماری نشان می‌دهد. این راهکار مقدار خطای حاصل، که ناشی از تبدیل معماری اعشاری به معماری ممیزثابت است را به میزان قابل توجهی بهینه می‌کند.



شکل ۱۰.۵: بلوک دیاگرام مربوط به تخمین تابع چندجمله‌ای

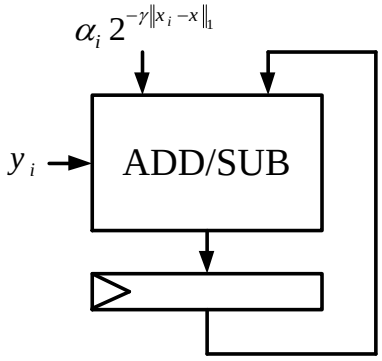
[۱] Multiply And Accumulate

ب **استفاده از الگوریتم CORDIC :** در صورتیکه هیچ ضرب کننده‌ای بر روی FPGA مورد استفاده‌ی کاربر وجود نداشته باشد و یا کاربر نخواهد از ضرب‌کننده‌های FPGA استفاده کند بهترین معماری برای محاسبه‌ی تابع کرنل، استفاده از الگوریتم CORDIC می‌باشد که در آن الگوریتم از هیچ‌گونه ضرب کننده‌ای استفاده نمی‌شود. برای آشنایی با اصول الگوریتم CORDIC می‌توانید اطلاعات کافی را در پیوست ب ببینید.

پ **استفاده از جداول جستجو:** ۲ معماری قبلی که تشریح شد از تکنیک تخمین زدن و تکنیک تکرارپذیر استفاده می‌کردند. یکی دیگر از معماری‌های قابل استفاده این است که مقدار کرنل را به ازای مقادیر مختلف محاسبه کرده و آن را در حافظه ذخیره کنیم وسپس در طول برنامه از این مقادیر ذخیره شده استفاده نماییم. پر واضح است که تعداد بیت‌های بیشتری در این روش استفاده خواهد شد زیرا می‌بایست مقدار کرنل را به ازای رنج وسیعی از مقادیر محاسبه شده و ذخیره گردند، در نتیجه بلوک‌های حافظه بیشتری مصرف خواهند شد. در FPGA هایی که مقدار حافظه‌ی داخلی بیشتری دارند و در نتیجه بلوک‌های RAM بیشتری برای ذخیره‌ی اطلاعات دارند، استفاده از این روش را در مواردی همچون تأخیر، ساده‌سازی و دقت از سایر موارد بهتر و بهینه‌تر می‌کند. در FPGA های ارزان قیمت، بلوک‌های حافظه کمتری در دسترس هستند لذا یا نباید از این روش استفاده نمود و یا اینکه باید از یک RAM خارجی برای استفاده از این معماری استفاده کرد که استفاده از RAM خارجی هم باعث ایجاد مقدار زمانی تأخیری در هنگام دسترسی CPU به حافظه می‌شود که مسلماً این مقدار زمان تأخیر نسبت به RAM داخلی (FLASH) بسیار بیشتر می‌باشد لذا استفاده از RAM خارجی در کنار FPGA های ارزان قیمت به منظور پیاده‌سازی این معماری از لحاظ سرعت عملکرد گزینه‌ی مناسبی نمی‌باشد.

ت **استفاده از کرنل سخت‌افزار پسند** در این پایان‌نامه، کرنلی را پیشنهاد نمودیم که می‌توان آن را تنها به کمک شیفت‌دادن و جمع‌کردن پیاده‌سازی نمود در حالیکه می‌دانیم این المان‌ها در یک قطعه FPGA بسیار ارزان قیمت نیز فراوان هستند. بنابراین در بین کرنل‌های پیشین بهترین و ساده‌ترین کرنل در صورتیکه نتیجه‌ی مطلوبی را نیز به همراه بیاورد، کرنل سخت‌افزار پسند می‌باشد.

۳ بر اساس رابطه‌ی تابع تصمیم که بارها در فصل تئوری به آن اشاره شد، اولین بلوکی که مورد استفاده قرار خواهد گرفت، بلوک جمع-تفریق کننده خواهد بود. که در آن بر اساس مقدار برچسب y_i مقدار جمع یا تفریق صورت می‌گیرد. همانطور که در شکل ۱۱.۵ هم نشان داده شده است:



شکل ۱۱.۵: بلوک دیاگرام محاسبه‌ی تابع تصمیم

برای به پایان رساندن پیاده‌سازی تابع تصمیم نیاز است تا یک سری داده به این هسته‌ی SVM اعمال شده و خروجی آن بر اساس اطلاعات وارد شده محاسبه گردد. برای اینکار از یک FPGA ارزان قیمت از خانواده‌ی Xilinx به عنوان مثال $Spartan - II$ استفاده خواهیم نمود. که این نوع از FPGA ها دارای منابع بسیار محدودی از ضرب کننده‌ها می‌باشند. خانواده‌ی دیگری که می‌توان روی آن پیاده‌سازی انجام داد خانواده‌ی $Spartan - 3$ می‌باشد. این خانواده را می‌توان به عنوان یک خانواده‌ای با سطح متوسط که دارای منابع محدود گیت‌ها و بلوک‌های محاسباتی است امّا نسبت به خانواده‌ی قبلی دارای منابع بیشتری است و سرانجام می‌توان بر روی یک برد با امکانات بسیار بیشتر از خانواده‌ای سطح بالا کار پیاده‌سازی را انجام داد که می‌توان به عنوان مثال به خانواده‌ی $Virtex - 4$ اشاره کرد. بنابر این باید ابتدا مشخص کرد که هدف ما چیست؟ آیا می‌خواهیم در مصرف منابع صرفه‌جویی کنیم و عملکرد مناسبی داشته باشیم؟ آیا می‌خواهیم به هر قیمتی تنها سرعت را بهینه کنیم؟ آیا تحقق این دو در کنار هم امکان‌پذیر است؟

قطعاً پاسخ منفی می‌باشد! همانطور که در شکل زیر هم نشان داده شده است، برای پیاده‌سازی الگوریتم محاسباتی نظیر آنچه بیان کردیم همواره یک معاوضه بین میزان منابع مصرفی و سرعت برقرار است. بنابراین فعلاً با دانش امروزی و با امکاناتی که در اختیار ما هست پیاده‌سازی الگوریتمی با حجم محاسباتی بالا که بتواند هم در مصرف منابع صرفه‌جویی کنیم و هم سرعت را تا حد امکان بالا ببرد، امکان‌پذیر نمی‌باشد. بالا بردن سرعت یعنی میزان مصرف منابع بیشتر و در نتیجه یعنی استفاده از خانواده‌ای سطح بالا و سرآخر یعنی قطعه‌ای گران قیمت تر! و اگر در مصرف منابع تا حد امکان صرفه‌جویی کنیم یعنی از قطعه‌ی ارزان قیمت‌تری استفاده کرده و در نهایت سرعت نسبت به حالت قبلی یقیناً کمتر خواهد بود.

شکل زیر نشان می‌دهد که برای هر نوع FPGA چه نوع الگوریتمی مناسب است در صورتیکه بخواهیم سرعت را فدای صرفه‌جویی در منابع کنیم و یا برعکس. که همانگونه که ملاحظه می‌کنید برای مثال اگر از

جدول ۱.۵: طبقه‌بندی الگوریتم مناسب جهت پیاده‌سازی کرنل انواع مختلف FPGA بر اساس نحوه‌ی عملکرد

نوع FPGA	صرفه‌جویی در منابع	بهینه کردن سرعت
SpartanII	تخمین چندجمله‌ای	CORDIC
SpartanIII	تخمین چندجمله‌ای	LUT
Virtex4	تخمین چندجمله‌ای	LUT

قطعه‌ی ارزان قیمت SpartanII استفاده می‌کنیم، در صورتیکه بخواهیم در مصرف منابع صرفه جویی کنیم باید از تخمین چندجمله‌ای استفاده کنیم و در صورتیکه بخواهیم سرعت را تا حداکثر ممکن بالا ببریم باید از الگوریتم CORDIC استفاده نماییم. در فصل بعد نتایج بررسی الگوریتم‌های فوق را بر روی یک تصویر آزمایش بیان خواهیم نمود در صورتیکه می‌دانیم تمامی مراحل آموزش از قبل به کمک نرم‌افزار MATLAB انجام شده و نتایج آن ذخیره شده است.

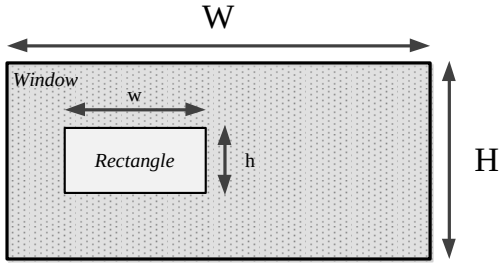
فصل ۶

نتایج و پیشنهادها

۱.۶ پیش پردازش لازم برای مرحله‌ی آزمایش PSVM

همانطور که گفته شد مرحله‌ی آموزش ماشین بردار پشتیبان می‌بایست به کمک نرم افزار و به طور جداگانه انجام شود و سپس از نتایج آن برای مرحله‌ی بعدی استفاده گردد. به همین منظور در این بخش به مرحله‌ی آموزش و نتایج آن می‌پردازیم.

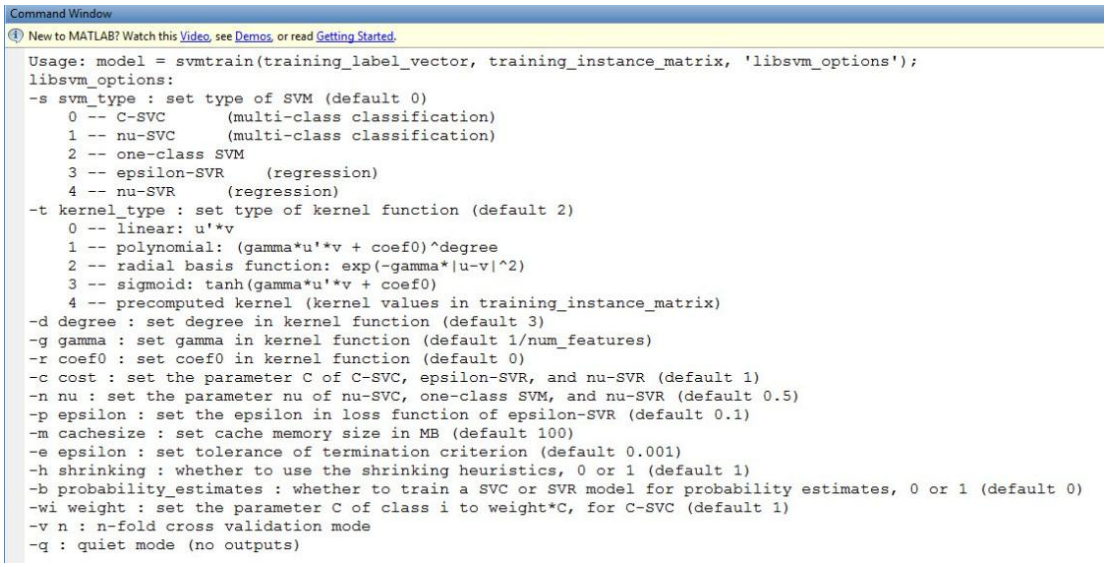
همانگونه که در فصل قبلی مراحل آماده‌سازی مجموعه داده‌ها و استخراج ویژگی‌ها بیان شدند ابتدا از مجموعه داده‌ها به تعداد ۲۰۰ نمونه برای نمونه‌های مثبت(چهره) و به تعداد ۴۰۰ نمونه هم برای نمونه‌های منفی(غیرچهره) انتخاب نمودیم. ابتدا از آنها ویژگی‌های هار را استخراج کردیم. از آنجا که نمونه‌های استفاده شده دارای ابعاد ۱۹×۱۹ بودند به ازای مجموع ۴ مستطیل هار بیان شده در فصل قبل و در شکل ۶.۵ به تعداد ۳۲۷۴۶ ویژگی استخراج شد. حال همانگونه که بیان شد می‌بایست از بین این نمونه‌ها بهترین نمونه‌ها را برای معرفی به ماشین بردار پشتیبان به کمک الگوریتم AdaBoost انتخاب نماییم. بدست آوردن تعداد ویژگی‌های هار استخراج شده به کمک شکل ۱.۶ و رابطه‌ی ۱.۶ بر آن بدست می‌آید.



شکل ۱.۶: طول و عرض مستطیل‌های هار در محاسبه‌ی تعداد ویژگی‌ها

$$\#ofFeatures = XY(W+1-w\frac{X+1}{2})(H+1-h\frac{Y+1}{2}) \quad (۱.۶)$$

که در آن $X = floor(\frac{W}{w})$ و $Y = floor(\frac{H}{h})$ می‌باشد. که از بین این تعداد ویژگی به کمک الگوریتم Adaboost و با ایجاد ۵۰ کلاسه‌بند ضعیف و ۲۰ طبقه برای هر طبقه کاسکود (که مدت زمان زیادی را هم صرف می‌کند و این از ضعف‌های این الگوریتم است) تنها ۳۶۱ ویژگی به عنوان ویژگی‌های برتر انتخاب شدند که می‌بایست به همراه برچسب هرکدام که شامل کدام دسته از نمونه‌ها می‌باشند (مثبت یا منفی) به عنوان ورودی به مرحله‌ی آموزش ماشین بردار پشتیبان معرفی گردند. از این به بعد با ماشین بردار پشتیبان مرحله‌ی آموزش را پیگیری خواهیم کرد. می‌توانید از آدرس http://www.csie.ntu.edu.tw/~cjlin/libsvm نرم افزار Libsvm را دانلود کرده و در صورتیکه بر روی کامپیوتر خود کامپایلر C/C++ را نصب داشته باشید می‌توانید از این نرم‌افزار استفاده نمایید. راهنمای چگونگی نصب این نرم‌افزار و استفاده از آن را می‌توانید در صفحه‌ی اینترنتی که در بالا آدرس آن داده شده است، بیابید. حال که ویژگی‌های منتخب و برچسب‌های مربوط به هرکدام را در اختیار داریم، می بایست آنها را در محیط MATLAB وارد فضای کاری خود نماییم. پس از این مرحله طبق الگوی نرم‌افزار Libsvm مرحله‌ی آموزش را انجام می‌دهیم. برنامه‌ی Libsvm پارامترهای زیادی دارد که هرکدام وظیفه‌ی خاصی را برعهده دارند و کاربر باید برای عملکرد درست آنها را به درستی تنظیم نماید و برخی از پارامترها نیز نیازی به تنظیم نداشته و تنها مقادیر حاصل خروجی را برمی‌گردانند. در شکل زیر این پارامترها را ملاحظه می‌کنید.



شکل ۲.۶: پارامترهای مورد نیاز برای اجرای نرم‌افزار Libsvm

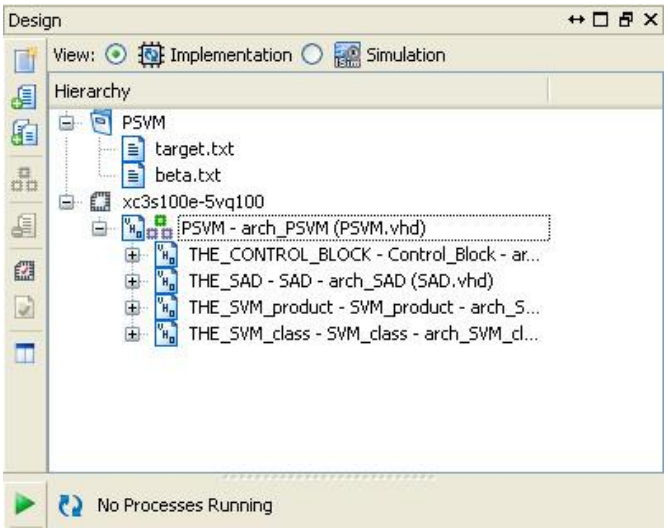
لازم به ذکر است که برای استفاده از Libsvm از آنجاییکه در این پایان نامه از کرنل سخت‌افزار پسند بهره خواهیم برد و این کرنل بصورت پیش‌فرض بر روی الگوریتم Libsvm قرار ندارد، لذا می‌بایست برای استفاده از آن، این کرنل را برای Libsvm تعریف کنیم. خوشبختانه این امکان در الگوریتم آن پیش‌بینی شده است و با انتخاب حالت چهارم از حالت‌های کرنل‌های Libsvm می‌توان کرنل تعریف شده‌ی خود را در این ابزار مورد استفاده قرار دهیم.

۲.۶ پیاده‌سازی الگوریتم PSVM به کمک کد VHDL

در این قسمت قصد داریم تا به صورت مختصر آنچه را که در فصل قبل به عنوان الگوریتم‌های معرفی شده در این پایان‌نامه بیان نمودیم را به زبان VHDL توضیح دهیم. همانطور که می‌دانید برنامه باید طوری نوشته شود که در درجه اول در کمترین میزان زمان ممکن اجرا شده و در مورد زبان‌های مرتبط با ماشین، کمترین حجم ممکن را اشغال نماید. زیرا منابعی که در FPGA ها به شدت کمبود آن احساس می‌شود، حافظه‌ی داخلی آنهاست و در درجه‌ی بعدی ساده باشد که هم نویسنده در حین نوشتن آن و هم کاربر در حین استفاده از آن دچار سردرگمی نشود. به همین دلیل، برنامه‌ی VHDL به گونه‌ای نوشته شده است که هر جزء[۱] در یک کد VHDL جداگانه نوشته شده است و سپس توسط جزء بالاتر خود صدا زده می‌شوند. در نهایت تمامی اجزاء توسط تابع اصلی برنامه که نام آن را PSVM گذاشته‌ایم صدا زده می‌شوند. در ادامه این اجزاء به طور مختصری مرور می‌شوند.

همانطور که در شکل زیر ملاحظه می‌کنید، تابع اصلی برنامه برنامه به نام PSVM شامل ۴ زیر تابع است که در حین اجرای برنامه آنها را صدا می‌کند. کامپایلر با دیدن این کد، ابتدا پایه‌های ورودی و خروجی کلی برنامه را ایجاد می‌کند و سپس زیرتابع‌ها را فراخوانی می‌کند.

[۱] Component



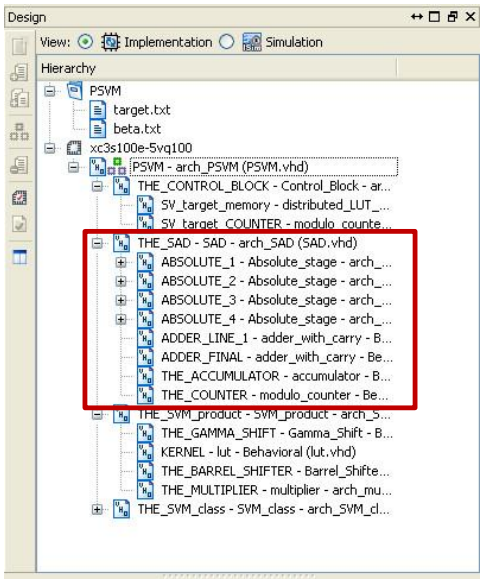
شکل ۳.۶: نمایش تابع اصلی برنامه که در محیط ISE نشان داده شده است

همانطور که در شکل زیر هم دیده می‌شود، زیربرنامه‌های تابع اصلی برنامه یعنی PSVM را ملاحظه می‌کنید.کامپایلر با دیدن تابع اصلی برنامه به سراغ اجزای آن رفته و **به صورت موازی** آنها را صدا زده و توسط سیگنال‌هایی که تعریف شده‌اند ارتباط بین آنها را برقرار کرده، خروجی هر کدام را به صورت موازی تولید کرده و به PSVM برگشته و مقادیر آنها را تحویل می‌دهد. در PSVM چهار جزء اصلی وجود دارند که آنها را در شکل ۴.۶ نشان دادیم. مهمترین آنها برای ما و برای این پایان نامه SAD می‌باشد که تئوری آن را در فصل ۵ توضیح دادیم. در اینجا نیز به صورت مختصری به کد VHDL نوشته شده در این زیربرنامه را بیان خواهیم کرد.

در اینجا باید خاطر نشان کرد که همانطور که در فصل ۵ هم بیان نمودیم، می‌خواهیم عملیات نرم مرتبه اول را به صورت موازی انجام دهیم. روند کار نیز به این صورت بود که ابتدا مقدار اختلاف $|A-B|$ را محاسبه می‌کنیم. برای این منظور برنامه باید ابتدا تشخیص دهد که آیا A از B بزرگتر است $(A > B)$ و یا خیر! در اینصورت علامت خروجی را تشخیص می‌دهد. در اینجا این کار را به کمک تولید بیت نقلی بررسی می‌کنیم. مطابق شکل ۹.۵ تا اینجا الگوریتمی را معرفی کردیم که ۲ مقدار را گرفته و نرم مرتبه اول آنها را محاسبه می‌کند. حال می‌خواهیم الگوریتم موازی سازی خود را بر روی این قسمت اعمال کنیم.

در مرحله‌ی بعدی و در کد نوشته شده ما داده‌ها را به ۴ دسته تقسیم نموده و مطابق شکل هردسته را جداگانه ولی بصور موازی وارد تابع SAD می‌کنیم و سپس مقدار تمامی دسته‌ها را با هم جمع می‌کنیم. همانطور که در شکل ۲.۶ هم ملاحظه می‌کنید تابع SAD نوشته شده دارای ۴ طبقه[۱] می‌باشد. اما بپردازیم

[۱] Stage



شکل ۴.۶: نمایش تابع اصلی برنامه که در محیط ISE با زیربرنامه های آن نشان داده شده است

به فرایند مهمی که تابع SAD آنها را انجام می‌دهد. اگر در فصل ۵ به شکل ۹.۵ نگاهی بیندازید ما در اینجا می‌خواهیم نحوه‌ی عملکرد آنها را به زبان VHDL درآوریم. لازم به ذکر است که صحت الگوریتم معرفی شده در شکل‌های مذکور در الگوریتم SAD مورد ادعای ما نمی‌باشد زیرا این الگوریتم بارها و بارها در مقالات متعدد و معتبر استفاده شده است و ما در این زمینه صحت عملکرد SAD را مطابق شکل ۹.۵ را بدون هیچ بحثی می‌پذیریم و تنها الگوریتم موازی سازی خود را به کمک آن پیاده نمودیم. در شکل زیر کد نوشته شده برای قسمت Entity را ملاحظه می‌کنید که در آن ۴ ورودی X و S به ترتیب بیان کننده‌ی بردارهای ورودی و بردارهای الگوی آموزش یافته یا همان X_i ها می‌باشند که طبق شکل ۱۰.۵ به عنوان ورودی های SAD می‌باشند. پورت‌های $VOUT$ و $Result$ هم به عنوان خروجی این جزء به اجزاءی که آن را صدا می‌زنند برگردانده می‌شود. همچنین سیگنال‌های کلاک و ریست هم برای کنترل شرایط ورود و خروج داده‌ها قرار داده شده‌اند که در اکثر پیاده‌سازی‌ها جزء پورت‌های مهم ورودی حضور دارند. در شکل ۶.۶ به عنوان مثال کد VHDL تولید کننده‌ی بیت نقلی را ملاحظه می‌کنید تا به کمک آن بتوانیم علامت خروجی نرم را تشخیص دهیم. سپس کافیست تا یک جمع کننده مقادیر خروجی را با هم جمع نماید تا حاصل نرم بدست آید. در واقع این ۲ تابع یعنی تولید کننده‌ی بیت نقلی و جمع‌کننده هر دو جزء زیرین تابع SAD می‌باشند و خود تابع SAD در تابع دیگری در لایه‌ی بالاتر صدا زده می‌شود.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6
7  entity SAD is
8
9     Port (
10              X1 : in std_logic_vector (11 downto 0);
11              X2 : in std_logic_vector (11 downto 0);
12              X3 : in std_logic_vector (11 downto 0);
13              X4 : in std_logic_vector (11 downto 0);
14
15              S1 : in std_logic_vector (11 downto 0);
16              S2 : in std_logic_vector (11 downto 0);
17              S3 : in std_logic_vector (11 downto 0);
18              S4 : in std_logic_vector (11 downto 0);
19
20              START : in  std_logic;
21              CLK : in  std_logic;
22              RESET : in  std_logic;
23              VOUT : out  std_logic;
24              RESULT : out  std_logic_vector (14 downto 0));
25
26 end SAD;
27
28
```

شکل ۵.۶: نمایش پورت‌های ورودی و خروجی تابع SAD

این ۲ تابع در تابع دیگری به نام Absolute صدا زده می‌شوند و مقدار آن به عنوان مقدار مقدار نرم دسته‌ی اول به SAD گزارش داده می‌شود! در SAD ۴ تابع Absolute قرار گرفته‌اند تا هر ۴ دسته از مقادیر را به صورت موازی بدست آورده و به لایه‌ی بالاتر گزارش نمایند. در شکل ۷.۶ کد VHDL برای تابع Absolute را ملاحظه می‌نمایید.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6
7  entity carry_generator is
8     Generic (   size : natural := 8 );
9     Port (   x : in  std_logic_vector ((size-1) downto 0);
10           y : in  std_logic_vector ((size-1) downto 0);
11           carry : out  std_logic;
12           carry_neg : out  std_logic);
13 end carry_generator;
14
15
16
17
18 architecture Behavioral of carry_generator is
19
20 signal temp : std_logic_vector(size downto 0);
21
22 begin
23
24    temp <= ('0' & x) + ('0' & y);
25    carry <= temp(size);
26    carry_neg <= not(temp(size));
27
28 end Behavioral;
```

شکل ۶.۶: زیر برنامه بیت نقلی از جمله زیر برنامه‌های تابع اصلی SAD

اگر شکل ۹.۵ را دوباره نمایش دهیم این بار مشخص می‌نماییم که چه قسمت‌هایی توسط تابع Absolute و چه قسمتی توسط تابع SAD پیاده‌سازی شده‌اند. کافیست به شکل ۸.۶ نگاهی بیاندازیم.

در نمام مراحل اجرای کدها در محیط ISE برنامه و توابع درونی آن به کمک یک ماشین حالت (*State Machine*) کنترل می‌شوند که این جزء از برنامه توسط کلاک کنترل می‌شود. در شکل ۹.۶ می‌توانید ماشین حالت استفاده شده و نحوه‌ی ارتباط آن با سایر قسمت‌های الگوریتم را ملاحظه نمایید.

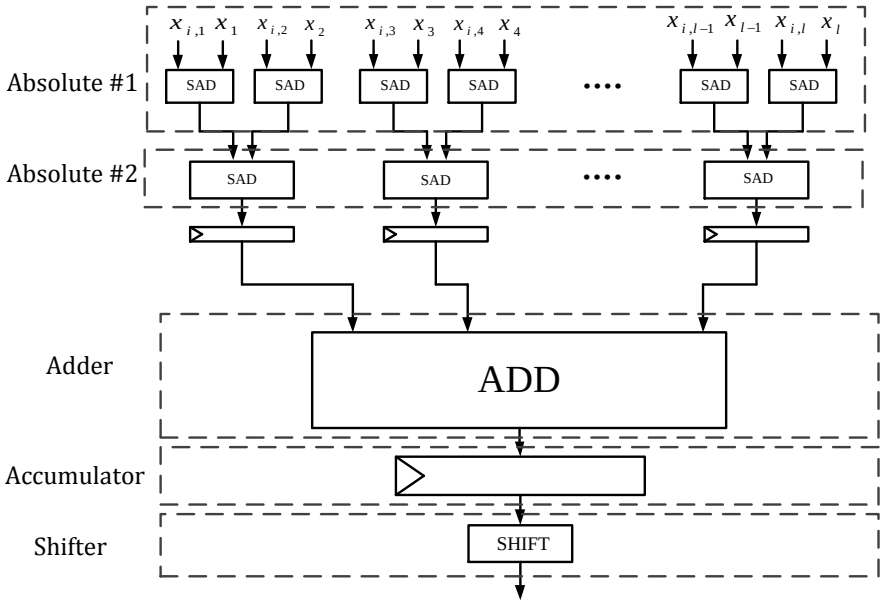
```
1  process (CARRY, CARRY_NEG, A, B)
2      begin
3
4          for i in (size-1) downto 0 loop
5
6              A_OUT(i) <= A(i) xor CARRY;
7              B_OUT(i) <= B(i) xor CARRY_NEG;
8
9          end loop;
10
11     end process;
12
13     process(CLK, RESET)
14     begin
15
16         if (RESET = '1') then
17
18             A_OUT_REGISTER <= (others => '0');
19             B_OUT_REGISTER <= (others => '0');
20
21         elsif (CLK'event AND CLK='1') then
22
23             A_OUT_REGISTER <= A_OUT;
24             B_OUT_REGISTER <= B_OUT;
25
26         end if;
27
28     end process;
```

شکل ۷.۶: نمایش قسمتی از کد VHDL نوشته شده برای تابع Absolute

۳.۶ نتایج

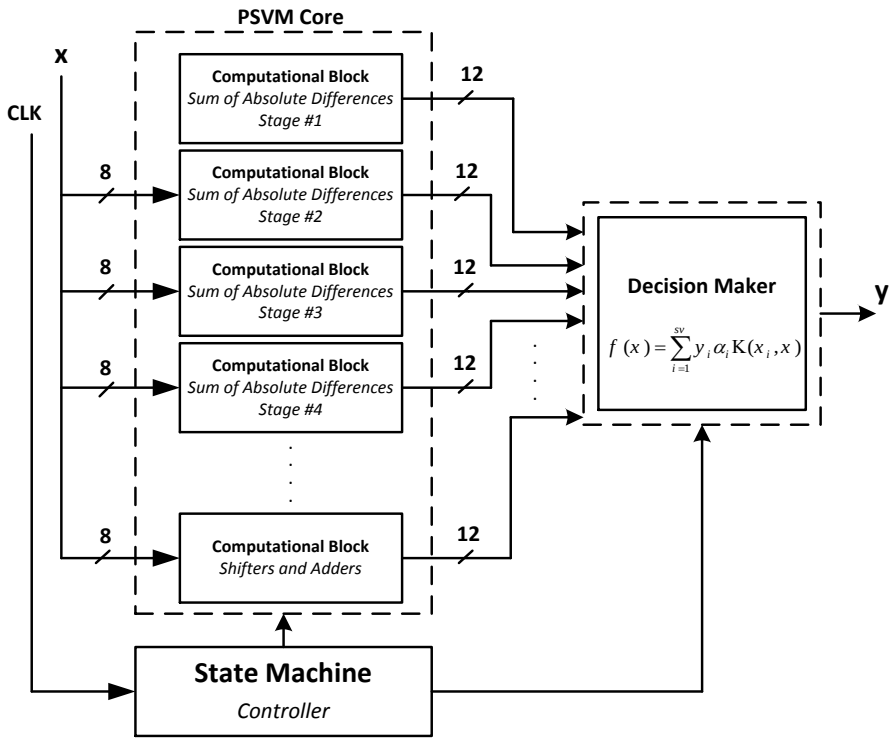
پس از کامل نمودن کدها و نوشتن TestBench و سپس اجرای آنها در نرم‌افزار ISE نتایج پیاده‌سازی و شبیه‌سازی PSVM را در این بخش ارائه خواهیم داد. در این بخش آمار و نتایج خود را با مرجع [۴۱] مقایسه خواهیم نمود و نشان خواهیم داد که الگوریتم مورد استفاده‌ی ما در این پایان‌نامه تا چه حد برتری به مرجع ذکر شده دارد. برای اینکار از FPGA ارزان قیمت و متوسطی به نام $Spartan3E$ از خانواده‌ی Xilinx استفاده می‌کنیم. برای انتخاب این قطعه در ISE مطابق شکل ۱۰.۶ عمل خواهیم نمود.

سپس مطابق شکل ۱۱.۶ وارد بخش شبیه‌سازی شده و با تهیه‌ی Testbench و تعریف پورت‌های ورودی برای شبیه‌ساز و همچنین سیگنال‌های Clock ، Start ، و Reset می‌توانیم پروژه را شبیه‌سازی کنیم.



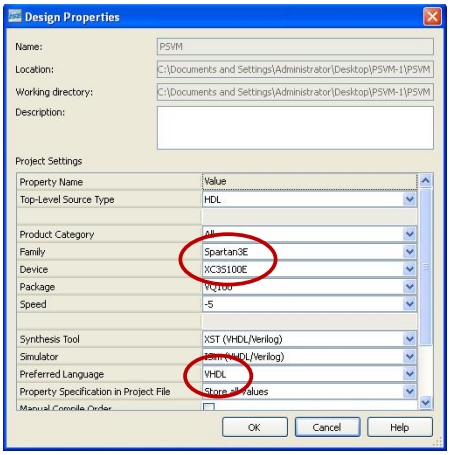
شکل ۸.۶: نمایش تابع اصلی برنامه که در محیط ISE با زیربرنامه های آن نشان داده شده است

سپس با شروع به شبیه‌سازی در صورتیکه خطایی و یا هشداری متوجه Testbench نباشد وارد محیط شبیه‌ساز ISim خواهیم شد. در صورتیکه در تنظیمات ISE تنظیم نماییم می‌توان از شبیه‌سازهای دیگر همچون شبیه‌ساز قدرتمند Modelsim هم استفاده نماییم.

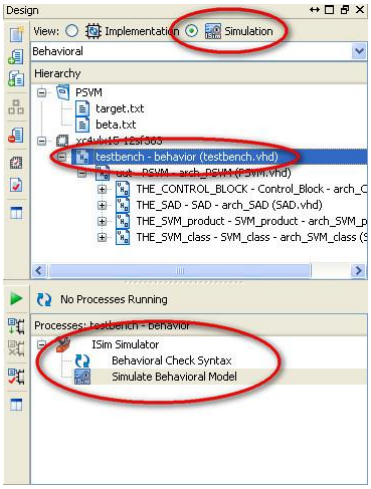


شکل ۹.۶: ماشین حالت بکار رفته در الگوریتم پیشنهادی

اگر بخواهیم مشخصات کلی سیستم خود را در یک جدول خلاصه کنیم جدولی همانند جدول زیر نشان دهنده‌ی مشخصات سخت‌افزاری و مقدار منابع مصرف شده را نشان می‌دهد. جدول فوق نشان می دهد که طراحی الگوریتم تا چه میزان بهینه می‌باشد و تنها میزان بسیار اندکی از منابع موجود مصرف شده‌اند که در مقایسه با مرجع [۴۱] کاملاً تغییرات محسوس و قابل ملاحظه‌ای دارد. در جداول زیر هم آماری از نحوه‌ی عملکرد سیستم و همچنین پارامترهایی را که در سیستم مورد استفاده قرار گرفته است را ملاحظه می‌نمایید. در این جدول‌ها نحوه‌ی عملکرد با توجه به نوع معماری استفاده شده در سیستم محاسبه شده است. در جدول ۳.۶ مقایسه‌ای بین سخت‌افزارهایی که ۲ آزمایش بر روی آنها صورت گرفته است، آورده شده است. در این جدول تفاوت بین نرم‌افزار MATLAB که بر روی کامپیوتر نصب بوده است با سخت‌افزار دوم یعنی FPGA آورده شده است. که در اینجا باید خاطر نشان کرد، علت افت دقت شناسایی در سخت‌افزار پیاده‌سازی شده استفاده از معماری ممیزثابت می‌باشد در حالیکه در نرم‌افزار کامپیوتری از معماری ممیزشناور استفاده می‌شود. که با توجه به محاسباتی که در مرجع [۲۱] آورده شده است مقدار برتری زمانی به اینصورت



شکل ۱۰.۶: انتخاب قطعه‌ی مورد نیاز از لیست قطعات در نرم افزار ISE



شکل ۱۱.۶: وارد شدن به مرحله‌ی شبیه‌ساز

محاسبه می‌گردد:

$$\mathbf{Times} = \frac{Time\ of\ LibSVM}{Time\ of\ PSVM} \qquad (۲.۶)$$

جدول ۱.۶: میزان منابع موجود و مصرف شده در پیاده سازی الگوریتم معرفی شده

لاجیک‌های موجود	تعداد موجود	تعداد استفاده شده	درصد استفاده
$SlicceFlipFlops$	12288	330	%2
4 $inputLUTs$	12288	325	%2
$OccupiedSlice$	6144	214	%3
$IOBs$	240	160	%66
$BUFG/BUFGCTRL$	32	1	%1
$FIFO16/RAMB1s$	48	3	%6
$DSP48s$	32	1	%3

جدول ۲.۶: پارامترهای بکار رفته در الگوریتم

پارامتر	مقدار
بردار ورودی	8 بیت
ضرایب	12 بیت
کرنل	12 بیت
پارامتر γ	2^{-1}

یعنی چیزی حدود ۱۳۳ برابر سریعتر از مقدار زمان محاسبه به کمک سخت‌افزارهایی که مشخصات آنها و تفاوت‌های آنها در جدول فوق آورده شده است. در رابطه با تأخیر محاسباتی در الگوریتم پیشنهاد داده شده باید خاطر نشان کرد که اگر از الگوریتم CORDIC استفاده نماییم بیشترین تأخیر متوجه سیستم خواهد شد و اگر از جداول جستجو استفاده نماییم کمترین تأخیر متوجه سیستم خواهد شد.
در جدول زیر این مقایسه را ملاحظه می‌نمایید.

نکته‌ای که در پایان باید به آن اشاره نمود این است که در صورتیکه بخواهیم سخت‌افزار موردنظر را بر روی یک برد آزمایشی پیاده‌سازی نماییم ملاحظات بسیار بیشتری را می‌بایست در نظر بگیریم. از جمله این ملاحظات مقدار حافظه‌ای است که باید برای چنین طراحی‌هایی پیش‌بینی کرد. پس از تهیه‌ی امکانات حافظه که بر روی بردهای آموزشی عموماً بصورت حافظه‌های موقتی Flash) (Memory ها می‌باشند، زمان دسترسی به این منابع و زمان جواب گویی این منابع به درخواست پروسسور از جمله دردسرهای ادامه‌ی کار می‌باشد. زیرا در این پایان نامه اطلاعات ورودی‌ها را از حافظه‌ی HardDisk کامپیوتر استخراج کردیم و به شبیه‌ساز اعلام نمودیم ولی اگر بخواهیم از حافظه‌های متصل شده به برد این اظلاعات را فراخوانی نماییم کار به مراتب سخت‌تر و ملاحظات آن به شدت افزایش پیدا می‌کند. به همین دلیل است که چون معمولاً FPGA حافظه‌هایی زیر ۱ مگابایت دارند، در کاربردهای تجاری و صنعتی از آنها در کنار یک میکروکنترلر استفاده می‌شود. آندسته از کارهایی که بلادرنگ بوده و حجم پردازشی خاصی را در زمان کوتاه می طلبد را به FPGA سپرده و بقیه‌ی وظایف را میکروکنترلر انجام می‌دهد.

جدول ۳.۶: تفاوت منابع، سرعت کلاک و دقت ۲ سخت‌افزار آزمایش شده

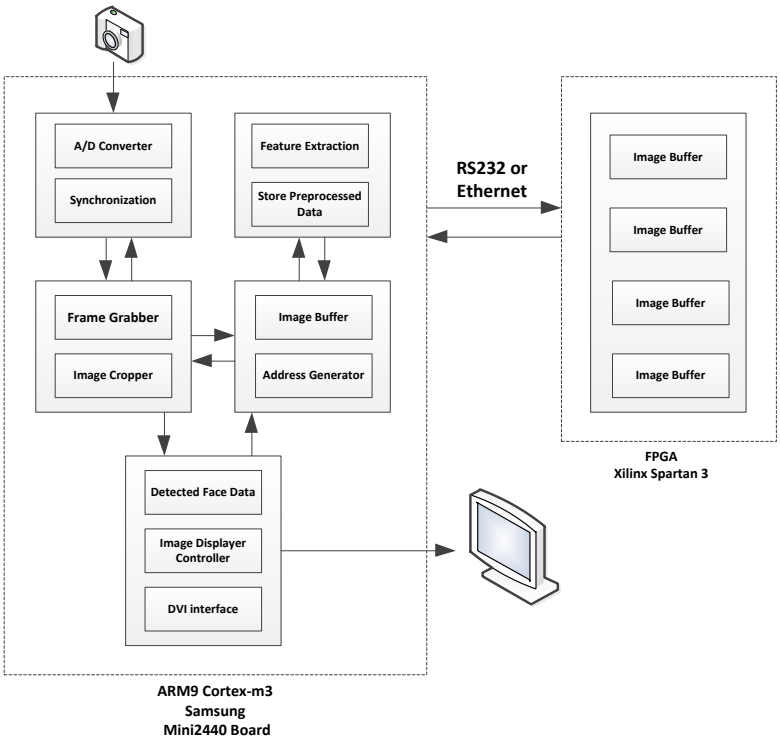
نام سخت‌افزار	مقدار حافظه	سرعت کلاک	دقت شناسایی	زمان پاسخگویی برای یک تصویر
$Spartan3E$	$Max\ 1MByte$	$100MHz$	%95	$30ms$
$IntelCOREi5$	$8GByte$	$2.4GHz$	%99	$4s$

جدول ۴.۶: تأخیر بدست آمده از کرنل‌های مختلف در مقایسه با PSVM

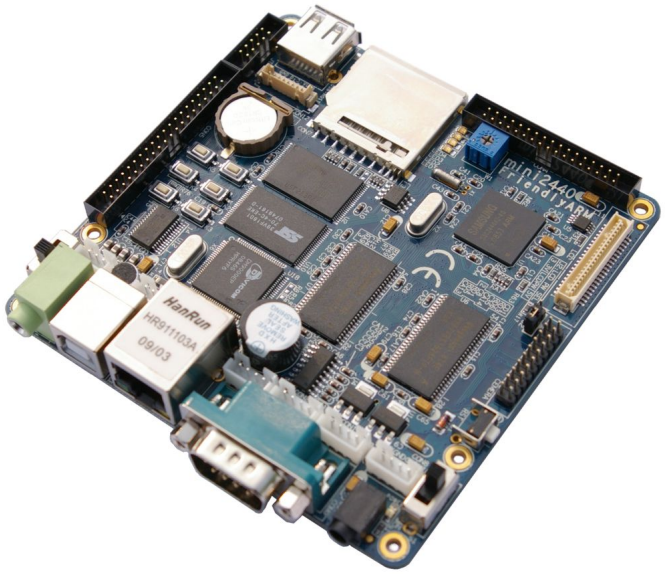
نوع کرنل	سرعت کلاک	زمان تأخیر
LUT	$240MHz$	$< 10ms$
CORDIC	$170MHz$	$< 15ms$
PSVM	$133MHz$	$< 30ms$

۴.۶ پیشنهادات برای کارهای آینده

در پایان این نوشتار می‌توان به بیان یکسری پیشنهادات برای راهکارهای آینده اشاره نمود. همانطور که مطالعه نمودید، این پایان‌نامه به بررسی پیاده‌سازی ماشین بردار پشتیبان موازی برای کاربرد تشخیص چهره پرداخت. به عنوان یک پیشنهاد می‌توان این را به یک مسئله‌ی بزرگتر و پیچیده‌تر بسط داد. مثلاً فرض کنید هدف ما تولید سخت‌افزاری است که بتواند پلاک خودروها را شناسایی نماید. همانطور که می‌دانید پلاک خودروها از ۱۰ رقم و ۳۲ حرف زبان فارسی تشکیل شده‌است. بنابراین چنین مسئله‌ای تشخیص یک مسئله‌ی چند کلاسه پیچیده است به طوریکه هم جنبه‌ی علمی را به خوبی ارضا می‌کند و هم جنبه‌ی کاربرد عملی دارد.
برای رشد و توسعه‌ی چنین سخت‌افزارهایی(خواهد تشخیص چهره باشد خواه تشخیص حروف و ارقام) می‌توان بردی را طراحی نمود که شامل یک دوربین CMOS ۳.۳ ولتی کوچک، میکروپروسسور FPGA و یک میکروکنترلر ARM به همراه ادوات و قطعات ضروری جانبی می‌باشد. میکروپروسسور تصاویر با اندازه‌ی کوچک و بسیار کم حجم را تهیه کرده و به میکروکنترلر می‌فرستد. میکروکنترلر ویدئو فریم‌های دریافتی را برش داده و به صورت مجموعه‌ای از تصاویر برای میکروپروسسورر ارسال می‌کند. میکروپروسسور که در اینجا FPGA می‌باشد وجود تصویر و یا عدم وجود آن را تشخیص داده و به میکروکنترلر اعلام می‌نماید که می‌تواند نتیجه را بر روی LCD نیز گزارش دهد. بدین ترتیب می‌توان این کار را برای پلاک خودروها نیز انجام داد. چنین پیاده‌سازی‌هایی به عنوان پیشنهاد ارائه در اینجا بیان شد زیرا هم جنبه‌ی عملی دارد و هم هسته‌ی آن محتوای درونی این پایان نامه را در بر می‌گیرد.
در شکل زیر به عنوان مثال طرحی مختصر از آنچه گفته شد آورده شده است:
که می‌تواند بر روی برد آموزشی زیر پیاده‌سازی گردد، همانطور که در شکل زیر هم نمایی از آن را ملاحظه می‌نمایید.



شکل ۱۲.۶: نمایی کلی از طراحی پشنهادی برای ادامه‌ی کار



شکل ۱۳.۶: برد آموزشی مناسب جهت پیشنهاد کارهای آینده

پیوست آ

مروری مختصر بر کلاسه‌بند
Adaptive Boosting

آ.۱ تئوری مقدماتی

فرض کنید مجموعه کاملی از ویژگی در اختیار داریم که برای محاسبه در کاربردهایی همانند شناسایی چهره، بسیار وسیع و کامل گردآوری شده باشند. حتی اگر این ویژگی‌ها، شامل ویژگی‌های بسیار ساده‌ای باشند به دلیل تعداد بسیار زیاد آنها محاسبات از لحاظ زمانی بسیار پرهزینه خواهد شد. بهترین راه حل برای برداشتن مشکل چنین محاسبات پیچیده و گسترده‌ای استفاده از یک کلاسه‌بند است بطوریکه بتواند یک مجموعه کوچکتری نسبت به مجموعه اول به ما تحویل دهد که شامل بهترین نوع جداسازی ویژگی‌های کلاس‌های مختلف از یکدیگر باشند. در اینصورت ما می‌توانیم به جای محاسبه مجموعه کامل ویژگی‌ها که از لحاظ وقت و هزینه مشکل‌ساز می‌باشد از مجموعه کوچکتری که خروجی کلاسه بندی شده می‌باشد استفاده نماییم. یکی از بهترین کلاسه‌بندها و الگوریتم‌ها که می‌تواند چنین کاری برای ما صورت دهد کلاسه‌بند AdaBoost می‌باشد. همانگونه که در شکل زیر هم ملاحظه می‌کنید الگوریتم AdaBoost ۲ هدف کلی زیر را دنبال می‌کند:

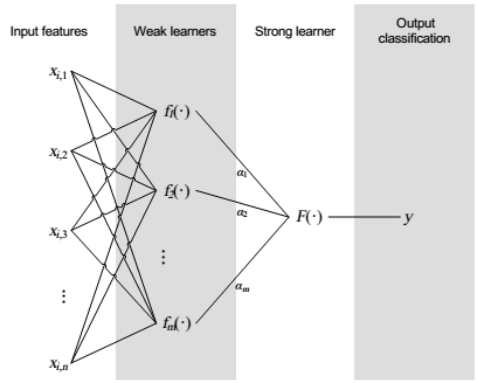
- انتخاب تعدادی ویژگی ($Weak\ Classifier$) از مجموعه کامل ویژگی‌ها که نماینده‌ی ویژگی‌های هر کلاس باشد.

- یادگیری یک کلاسه‌بند قوی ($Strong$) که ترکیب خطی از ویژگی‌های انتخاب شده در مرحله‌ی قبل می‌باشد.

طرح اولیه از یک کلاسه‌بند AdaBoost را در زیر ملاحظه می‌نمایید.

شکل آ.۱: طرح اولیه کلاسه‌بند AdaBoost

همچنین در شکل زیر نحوه‌ی عملکرد این طرح اولیه را به عنوان نمونه‌ای از شبکه‌های عصبی ملاحظه می نمایید:



شکل آ.۲: AdaBoost به عنوان شبکه عصبی

در ادامه توضیح داده خواهد شد که ما از این کلاسه‌بند به دلیل کارآیی و نتایج بسیار خوب آن به جای کلاسه‌بندهای مشابه استفاده خواهیم نمود.

فرض کنید یک مجموعه‌ای از ویژگی‌های استخراج شده در دست دارید و می‌خواهیم کلاسه‌بندی را بر روی ویژگی‌ها اعمال کنیم تا کلاس‌ها از یکدیگر تفکیک شده و شبکه بر روی آنها آموزش ببیند. تعداد مختلفی کلاسه‌بند وجود دارد که می‌توانند این کار را انجام دهند و روی مجموعه داده‌ها آموزش ببیند برای مثال:

* ماشین بردار پشتیبان
* ترکیبی از مدل‌های گوسی
* استفاده از ویژگی‌های خام در شبکه‌های عصبی ساده مانند MLP

آ.۲ چرا AdaBoost ؟

AdaBoost یکی از بهینه‌ترین الگوریتم‌های Boosting که ترکیبی از آموزنده‌های آماری خطی می‌باشد که نه تنها خطای یادگیری را تا حد بسیار چشمگیری کاهش می‌دهد بلکه خطای عمومی[۱] را نیز به صورت بسیار زیادی پایین می آورد. همانند همه‌ی توابع آموزشی، AdaBoost هم مزایا و هم معایبی دارد که در زیر به آنها اشاره می‌کنیم.

مزایا:

۱ همانگونه که قبلاً در شکل آ.۱ هم ملاحظه شد، AdaBoost الگوریتمی است که ۲ ورودی دارد: یکی مجموعه داده‌های آموزشی و دیگری مجموعه‌ای از ویژگی‌ها. بنابراین در این الگوریتم هیچ گونه نیازی نمی‌باشد تا اطلاعاتی از ساختار صورت را به الگوریتم بدهیم. بنابراین مهمترین ویژگی‌هایی که نماینده‌ی آن کلاس می‌باشد به صورت خروجی این الگوریتم بدست خواهد آمد.

۲ در هر مرحله از آموزش، نمونه‌های مثبت و منفی بوسیله‌ی کلاسه‌بندی که درهمان مرحله ساخته شده است، آزمایش می‌شود. اگر یم نمونه مثلاً x_i به صورت اشتباه کلاسه‌بندی گردد به این معنی است که این نمونه از آن جمله نمونه‌هایی است که کلاسه‌بندی را دچار دردسر می‌کند و بنابراین ممکن است این نمونه به خوبی و درستی به کلاسی که به آن تعلق دارد اختصاص داده نشود. جهت افزایش قدرت الگوریتم در مورد این نمونه‌های مشکل ساز به ایت صورت عمل می‌شود که نمونه‌هایی که به سادگی و بدون مشکل کلاسه‌بندی شده‌اند و مشکلی ندارند دارای وزن کمتر و نمونه‌هایی که مشکل ساز بوده‌اند و بخوبی کلاسه‌بندی نشده‌اند دارای وزن بیشتر در مرحله‌ی بعدی آموزش می‌باشند تا در مراحل بعدی تمرکز بیشتری بر روی آن ها جهت کلاسه‌بندی بهتر صورت گیرد.

۳ خطای آموزش از لحاظ تئوری، به صورت نمایی به صفر همگرا می‌شود. همانگونه که در [۴۲] بحث شده است با دادن یک مجموعه محدود از نمونه‌های مثبت و منفی در نهایت با تعداد محدودی از تکرار، خطای آموزش به صفر میل می‌کند.

معایب:

۱ نتایج نهایی به طور کامل به مجموعه داده‌ها و $Weak\ Classifier$ ها بستگی دارد. کیفیت نتیجه‌ی نهایی کلاسه‌بند به شدت تحت تأثیر مجموعه‌ی داده‌های آموزش و مجموعه ویژگی‌ها است.

[۱] Generalization Error

۲ در هر مرحله‌ی تکرار، الگوریتم همه‌ی ویژگی‌ها را آزمایش می‌کند که نیازمند صرف زمانی است که کاملاً نسبت مستقیمی با اندازه مجموعه داده‌ها دارد. فرض کنید مجموعه داده‌ها شامل هزاران نمونه‌ی مثبت و منفی باشد. در اینصورت در هر مرحله‌ی تکرار از مراحل آموزش زمان بسیار زیادی صرف انجام محاسبات آموزش می‌گردد.

آ.۳ AdaBoost

AdaBoost یکی از روش‌های معروف و محبوب بالاتر بردن دقت تکنیک یادگیری با نظارت می‌باشد. این روش شامل ۲ بخش وزن‌دهی تکراری و ترکیب کردن کلاسه‌بندهای ضعیف و بوجود آوردن یک کلاسه بند قوی می‌باشد که دقت را تا حد بسیار بالایی افزایش می‌دهد.

اولین پیشوایان در تئوری AdaBoost آقایان فروید و شاپیر بودند [۴۳] و امروزه علاقه‌مندی‌های زیادی در مورد AdaBoost در زمینه‌ی یادگیری ماشین بوجود آمده است و همانطور که در مقالات علمی هم مشاهده می‌شود در زمینه‌ی پردازش تصویر [۴۴] ، واکاوی داده‌ها [۴۵] مورد استفاده قرار گرفته است. در نرم افزار MATLAB نیز امکاناتی فراهم شده است که به کاربران اجازه‌ی استفاده راحت از این الگوریتم را در قالب یک جعبه افزار می‌دهد.

در واقع Adaboost یک متا الگوریتم است که به منظور ارتقاء عملکرد، همراه دیگر الگوریتم‌های یادگیری استفاده می‌شود که در اینجا در کنار الگوریتم ماشین بردار پشتیبان استفاده خواهد شد. در این الگوریتم، طبقه‌بند در هز مرحله‌ی جدید به نفع نمونه‌های به اشتباه طبقه‌بندی شده در مرحله‌ی قبل تنظیم می‌گردد. Adaboost نسبت به داده‌های نویزی و پرت بسیار حساس است و نسبت به مشکل بیش برازش از بیشتر الگوریتم‌های یادگیری برتری دارد. طبقه‌بند پایه که در اینجا استفاده می‌شود تنها کافیست از طبقه‌بند تصادفی (50%) بهتر باشد و به این ترتیب بهبود عملکرد الگوریتم با تکرارهای بیشتر بهبود می‌یابد. حتی کلاسه‌بندهای با خطای بیشتر از حالت تصادفی با گرفتن ضریب منفی عملکرد کلی را بهبود می‌بخشند.

در الگوریتم Adaboost در هر بار تکرار یک کلاسه‌بند ضعیف اضافه می‌شود و در هر بار فراخوانی بر اساس اهمیت نمونه‌ها، وزن های D_t بروزرسانی می‌گردند.در هر تکرار وزن نمونه‌های به اشتباه طبقه‌بندی شده افزایش و وزن نمونه‌های به درستی طبقه‌بندی شده کاهش می‌یابد. بنابراین طبقه‌بند جدید تمرکز بر روی نمونه‌هایی که سخت‌تر یادگرفته شده‌اند، خواهد داشت.

الگوریتم ۱ الگوریتم AdaBoost [۴۶]

1 **Input:** $S = (x_1, y_1), \ldots, (x_N, y_N)$ Number of iterations T.

2 **Initialize:** $d_n^{(1)} = 1/N$ for all n=1,...,N

3 **Do for** t=1,...,T,

(i) Train classifier with respect to the wighted sample set $\{S, d^{(t)}\}$ and obtain hypothesis $h_t : x \mapsto \{-1, +1\}$, *i.e.* $h_t = L(S, d^{(t)})$

(ii) Calculate thw wieghted training error ϵ_t of h_t:

$$\epsilon_t = \sum_{n=1}^{N} d_n^{(t)} \mathbf{I}(y_n \neq h_t(x)_n)$$

(iii) Set:

$$\alpha_t = \frac{1}{2} \log \frac{1 - \epsilon_t}{\epsilon_t}$$

(iv) Update the wieghts:

$$d_n^{(t+1)} = d_n^{(t)} \exp\{-\alpha_t y_n h_t(x_n)/Z_t\}$$

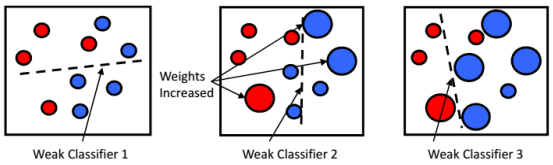
where Z_t is a normalization constant, such that $\sum_{n=1}^{N} d_n^{(t+1)} = 1$.

4 **Break if** $\epsilon_t = 0$ or $\epsilon_t \geq \frac{1}{2}$ and set T=t-1.

5 **Output:** $f_T(x) = \sum_{t=1}^{T} \frac{\alpha_t}{\sum_{r=1}^{T} \alpha_r} h_t(x)$

که در آن Z_t یک عامل نرمالیزاسیون با مقدار $\sum_i D_t(i) exp(-\alpha_t y_i h_t(x_i))$ است که موجب می‌شود D_{t+1} یک توزیع احتمالی مجاز را نشان دهد(مجموع روی x ها یک شود).

بنابراین بعد از انتخاب بهینه کلاسه‌بند h_t برای توزیع D_t آن دسته از نمونه‌ها که طبقه‌بند آنها را غلط طبقه‌بندی کرده است، وزن بیشتری نسبت به بقیه داده می‌شود. بنابراین وقتی الگوریتم طبقه‌بندها را بر اساس توزیع D_{t+1} آزمایش می‌کند، طبقه‌بندی انتخاب می‌گردد که نمونه‌های غلط طبقه‌بندی شده را بیشتر تشخیص دهد. که در بخش محاسبات نرم‌افزاری این قسمت پیاده سازی شده و بر روی مجموعه ویژگی‌های هار که از مجموعه داده‌های برگزیده اعمال گردید. برای پیاده‌سازی الگوریتم فوق بصورت کاسکود از ۵۰ کلاسیفایر ضعیف و ۲۰ طبقه‌ی کاسکود استفاده شده است.



شکل آ.۳: نمونه‌ای از نحوه‌ی عملکرد الگوریتم AdaBoost [۴۷]

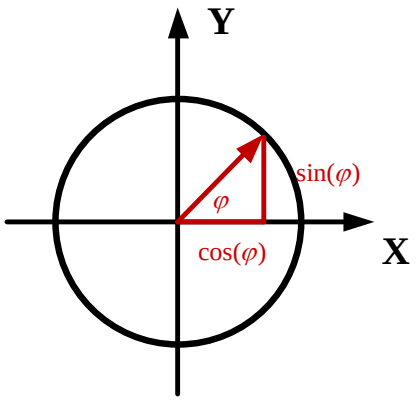
مثالی کوچک از چگونه عمل کردن این الگوریتم را در شکل آ.۳ ملاحظه می‌کنید. در این مثال نشان داده است که الگوریتم فوق با چند تکرار و با نسبت دادن وزن‌های کمتر به نمونه‌های درست کلاسه‌بندی شده و وزن‌های بیشتر به نمونه‌های غلط کلاسه‌بندی شده درن نهایت به بهترین شکل ۲ کلاس قرمز و آبی را از یکدیگر تفکیک می‌کند. شایان ذکر است که دیگر شبکه‌های عصبی موجود قادر نمی‌باشند به خوبی AdaBoost و به این شکل ۲ کلاس را از یکدیگر تفکیک کنند.

پیوست ب

اصول الگوریتم CORDIC

ب.۱ تئوری مقدماتی

فرض کنید نقطه (۱،۰) روی محورهای مختصات به اندازه ϕ درجه در جهت خلاف حرکت عقربه ساعت دوران کند. در این صورت محورهای مختصات بردار دوران یافته جدید (x،y) به صورت زیر بدست خواهد آمد.



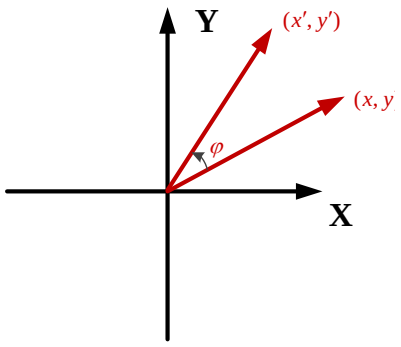
شکل ب.۱: نمایش مختصات بردار دوران یافته جدید

که در شکل فوق داریم:

$$x = cos\phi$$

$$y = sin\phi$$

حال فرض کنید بردار (x،y) بدست آمده در مرحله قبل را این بار به اندازه ϕ درجه دوران دهیم.
مختصات بردار جدید حاصل از این دوران برای هر (x،y) به صورت زیر خواهد بود:



شکل ب.۲: دوران یافته مختصات قبلی به اندازه ϕ درجه

$$x' = xcos\phi - ysin\phi$$

$$x' = ycos\phi + xsin\phi$$

در واقع این روش، روش سنتی پیاده‌سازی دوران برداری ۲ بعدی است که در آن (x،y) مختصات ابتدایی و (x'،y') مختصات نهایی بردار مورد نظر است. برای پیاده‌سازی سخت‌افزاری این معادلات به چهارضرب کننده و دو ضرب و جمع کننده و همچنین دسترسی به جدول مراجعه (LUT) برای محاسبه ضرایب مثلثاتی نیاز است. الگوریتم CORDIC دوران ۲ بعدی را با بکارگیری معادلات بازگشتی و استفاده از فقط عملیات شیفت و جمع محاسبه می‌کند.
برای شروع کار، معادله فوق را می‌توان به صورت زیر بازنویسی کرد:

$$x' = cos\phi[x - ytan\phi]$$

$$y' = cos\phi[y + xtan\phi]$$

اگر زوایای دوران محدود شود به نحوی که $tan\phi = \pm 2^{-i}$ ، آنگاه پیاده‌سازی آن با استفاده از جمله tan به یک عملیات شیفت ساده، کاهش پیدا می‌کند. این اصول کار الگوریتم CORDIC است. یعنی دوران برداری به صورت سلسله‌ای از دوران‌های پی‌درپی کوچکتر هر کدام با زاویه $tan^{-1}(2^{-i})$ به فرم زیر درمی آید.

$$x_{i+1} = K_i[x_i - y_i d_i 2^{-i}]$$

$$y_{i+1} = K_i[y_i - x_i d_i 2^{-i}]$$

که در آن

$$K_i = cos(tan^{-1}(2^{-i})) = \frac{1}{\sqrt{1+2^{-2i}}}$$

$$d_i = \pm 1$$

در معادله فوق اگر ثابت K حذف شود، الگوریتم دوران برداری فقط شامل شیفت و جمع خواهد شد. ضرب K_i ها را می‌توان در جای دیگری از سیستم انجام داد و یا اینکه آن را به عنوان بخشی از بهره کل سیستم در نظر گرفت. ضرب K_i ها به صورت زیر تعریف می‌کنیم:

$$K = \prod_{i=0}^{n} K_i$$

اگر تعداد تکرارها به سمت بینهایت میل کند، مقدار K تقریباً برابر ۰/۶۰۷۳ خواهد شد. بنابراین می‌توان این عدد ثابت را فقط یکبار در مرحله آخر در بهره سیستم ضرب کرد. الگوریتم CORDIC در ۲ حالت کار می‌کند: در حالت اول که مد دوران نام دارد، بردار ورودی توسط یک زاویه مشخص دوران می‌کند. در حالت دوم که مد برداری نام دارد، بردار ورودی به سمت محور x دوران می‌کند و زاویه مورد نیاز برای این دوران ذخیره می‌شود.

ب.۲ مُد دوران

برای مد دوران روابط الگوریتم CORDIC بصورت زیر در می آید:

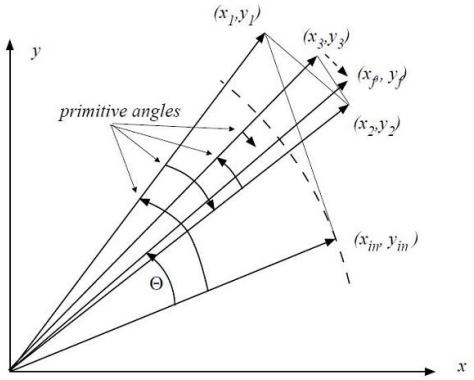
$$\mathbf{x}_{i+1} = \mathbf{x}_i - \mathbf{y}_i \mathbf{d}_i 2^{-i}$$
$$\mathbf{y}_{i+1} = \mathbf{y}_i + \mathbf{x}_i \mathbf{d}_i 2^{-i}$$
$$\mathbf{z}_{i+1} = \mathbf{z}_i - \mathbf{d}_i tan^{-1}(2^{-i})$$

که در آن :

$$d_i = \begin{cases} -1 & ,if \quad z_i < 0 \\ +1 & , \quad ow \end{cases}$$

اگر روابط بالا n بار تکرار شوند، مقادیر نهایی زیر را نتیجه می‌دهد:

$$
\begin{aligned}
x_f &= A_n[x_0\cos(z_0) - y_0\sin(z_0)] \\
y_f &= A_n[y_0\cos(z_0) + x_0\sin(z_0)] \\
z_f &= 0 \\
A_n &= \prod_n \sqrt{1+2^{-2i}}
\end{aligned}
$$



شکل ب.۳: چرخش بردارهای (x_{in}, y_{in}) و رسیدن به بردارهای (x_f, y_f)

این روابط شکل زیر را نشان می‌دهد که نتیجه‌ی مد دورانی الگوریتم CORDIC می‌باشد.

ب.۳ مُد برداری

در مد برداری روابط الگوریتم CORDIC به همان صورت قبل است اما شرایط تغییر علامت تابع تصمیم d_i به صورت زیر تغییر می‌کند:

$$
d_i = \begin{cases} -1 & ,if \quad y_i < 0 \\ +1 & , \quad ow \end{cases}
$$

بنابر این مقادیر نهایی به صورت زیر بدست می‌آید:

$$
\begin{aligned}
x_f &= A_n\sqrt{x_0^2 + y_0^2} \\
y_f &= 0 \\
z_f &= z_0 + tan^{-1}(\frac{y_0}{x_0}) \\
A_n &= \prod_n \sqrt{1+2^{-2i}}
\end{aligned}
$$

پیوست پ

مروری بر راهکارهای تشخیص چهره در تصاویر

پ.۱ مقدمه

همانگونه که در اهداف پایان نامه قیدگردیده بود، می‌بایست مرحله‌ی آموزش ماشین بردار پشتیبان را توسط نرم افزار کامپیوتری مانند MATLAB انجام داد و تنها مرحله‌ی آزمایش آن را بر روی سخت افزار پیاده‌سازی کرد زیرا اجرای مرحله‌ی آموزش بر روی سخت افزار برای کاربردهایی که مورد نظر ما هستند، هم از لحاظ مصرف حجم زیادی از فضای سخت افزار و هم از لحاظ پیچیدگی و مشکل بودن پیاده‌سازی توابع پیچیده‌ی ریاضی، مقرون به صرفه نیست.

در اینجا ما از نرم افزار MATLAB و برنامه‌ی LibSVM که برای ماشین‌های بردار پشتیبان نوشته شده است، استفاده خواهیم کرد. این برنامه سال‌ها در پایان نامه‌ها و مقالات مختلفی استفاده شده است و دارای صلاحیت بسیار بالایی است و می‌توان بخوبی و با اطمینان بالایی از آن برای اجرای مرحله‌ی آموزش استفاده کرد. مرجع این نرم افزار در مرجع [۲۱] آورده شده است که می توان بصورت رایگان آن را بارگیری کرده و از آن استفاده کنید.

بنابر این برای شروع ابتدا به کمک برنامه‌ی LibSVM ، مرحله‌ی آموزش را اجرا می‌کنیم و پس از اجرای آن پارامترهای لازم را استخراج کرده و از آنها برای استفاده در مراحل بعدی بهره می‌بریم. همانگونه که در فصل دوم پایان نامه در قسمت تئوری ماشین‌های بردار پشتیبان هم اشاره کردیم، پارامترهایی نظیر پارامتر C در بحث تئوری وجود دارند که می‌باید بصورت تجربی مقدار بهینه‌ی آن را بدست آورد و یا باید از الگوریتم‌هایی نظیر الگوریتم ژنتیک و یا الگوریتم PSO استفاده کرد و بهترین مقدار این پارامترها را بدست آورد. در اینجا ما بصورت تجربی و با کمک سعی و خطا بهترین مقدار پارامترها را بدست می‌آوریم تا نتیجه‌ی حاصل از ماشین بردار پشتیبان بر روی فرایند تشخیص چهره با کمترین خطای ممکن صورت گیرد و سپس از نتایج

آن برای مرحله‌ی بعد بر روی سخت افزار استفاده خواهیم نمود.
در این فصل مقدمه‌ای بر سیستم تشخیص چهره ارائه خواهیم داد، سپس مرحله‌ی آموزش را توسط نرم افزار بر روی داده‌های خود اجرا کرده و نتایج آن را برای مرحله‌ی آزمایش استفاده خواهیم کرد. سپس الگوریتم مورد نظر این پایان نامه را شرح داده و روند پیاده‌سازی سخت افزاری را مطرح خواهیم کرد.

پ.۲ ساختار سیستم تشخیص چهره

با پیشرفت سیستم‌های خودکار در زندگی امروزه‌ی ما، تعداد ابزارهای کاربردی از جمله سیستم‌های تشخیص چهره و آنالیز روز به روز در حال گسترش و توسعه می‌باشند.
تشخیص چهره[۱] ، ردیابی چهره[۲] ، حالت چهره[۳] و ... از جمله کاربردهای این ابزارها می‌باشند. تشخیص چهره شامل تشخیص هویت افراد از روی تصویر چهره‌ی آنها می‌باشد. مثلاً اینکه با دیدن چهره‌ی شخص سیستم بصورت خودکار تشخیص دهد این تصویر مربوط به یک شخص آسیای شرقی است یا یک شخص با رنگ پوست سفید یا سیاه، نوع مرد یا زن، جوان یا پیر و... است. این اطلاعات می‌تواند شامل نام افراد هم شود که می‌توان از این کاربردها در سرویس‌های امنیتی استفاده کرد. یافتن چهره در واقع پیدا کردن تصویر چهره‌ی یک شخص در مکان مشخصی از تصویر می‌باشد و تشخیص دادن آن فرد و دادن اطلاعات فرد به سرویس اطلاعاتی که به آن متصل است و از آن پیروی می‌کند می‌باشد. ردیابی چهره، معمولاً در یک رشته از تصاویر و یا به بیان دیگر در تصاویر ویدئویی می‌باشد که در آنها یک رشته‌ی عظیمی از تصاویر در ثانیه وارد سیستم شده و سیستم در این رشته تصاویر دنبال چهره‌ی فرد می‌گردد، آن را پیدا کرده و در رشته‌های تصاویری که مدام در حال بروزرسانی می‌باشند اطلاعات مکانی چهره فرد را پیگیری می‌کند. از این کاربرد می‌توان در سیستم امنیتی بانک‌ها، مدارس، سرویس‌های اطلاعاتی و... که از سیستم مانیتورینگ ویدئویی استفاده می‌کنند، قابل استفاده است.
یافتن حالت چهره، از جمله ابزارهایی است که حالت‌های مختلفی را از چهره‌ی شخص به سیستم گزارش می‌دهد از قبیل شادی، هیجان غم و غیره. بنابر گفته‌های قبل، اولین قدم برای تمامی کاربردهای ذکر شده این است که بتوانیم چهره را در یک تصویر تشخیص دهیم. بنابراین یافتن چهره‌ی انسان در یک تصویر به عنوان اولین قدم و اساسی ترین قدم مورد نیاز تمامی ابزارهای فوق می‌باشد. ما نیز به دلیل اهمیت این مرحله، برای معرفی یک کابرد مفید جهت ارائه‌ی این پایان نامه، یافتن چهره‌ی شخص در یک تصویر را به عنوان کاربرد انتخاب کرده و سعی خواهیم کرد تا اهداف سخت افزاری و نرم افزاری خود را در این راستا

[۱] Face recognition
[۲] Face Localization
[۳] Facial Expression

دنبال کنیم.

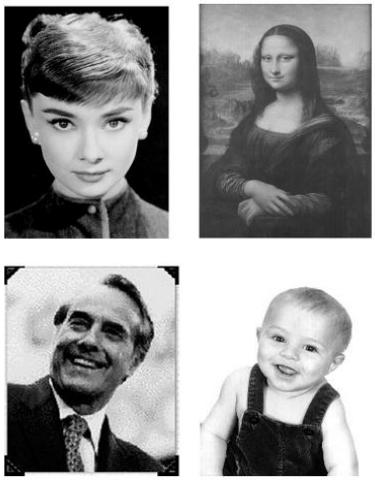
یافتن چهره در تصاویر اولین بار در سال ۱۹۷۰ میلادی در جریان تشخیص چهره‌ی افراد در گذرنامه‌های آنها به عنوان یک ایده مطرح شد. در اوایل دهه‌ی ۱۹۹۰ میلادی تکنیک‌های مختلفی جهت انجام این ایده مطرح و به بحث گذاشته شد. این ایده آنقدر مورد توجه و حمایت دانشمندان و مهندسین قرار گرفت که تا امروزه برای آن بیش از ۱۵۰ روش و تکنیک مختلف در مقالات متعدد ارائه شده است.

پ.۱.۲ مشکلات روش یافتن چهره در تصاویر

دلیل اینکه در ایجاد سیستم‌های خودکار هوش مصنوعی تأخیر حاصل شد، این بود که در بوجود آمدن و رشد و گسترش شبکه‌های عصبی مدت زیادی تأخیر بوجود آمد. پس از مدتی تأخیر در شبکه‌های عصبی سرانجام کلاسه‌بندها به رشد و کمال رسیدند و این ایده که یافتن چهره در تصاویر و کلاسه‌بندی آن دوباره از سرگرفته شود، قوت گرفت در حالیکه تشخیص چهره به کمک چشم یک اپراتور بسیار سریع و به سهولت انجام می‌گرفت لذا انجام این کار رو به یک اپراتور می‌سپردند تا یک سیستم مانیتورینگ خودکار. همچنین تصویر چهره‌ی انسان به دلیل ویژگی‌هایش از شناسایی اشیأ مانند هواپیما و خودرو دشوارتر است. زیرا چهره‌ی انسان‌ها دامنه‌ی تغییرات زیادی را شامل می‌شود از جمله نحوه‌ی چشم‌ها، گوش، بینی، ابروها در حالت‌های مختلف از جمله مانند ترس، نگرانی، شادی و همچنین حالت قرار گرفتن صورت، زاویه‌ی نور تابیده شده و دیگر عوامل که باعث می‌شود شناسایی چهره‌ی انسان سخت و دشوارتر از اشیأ باشد. در شکل زیر تصویر چند چهره را ملاحظه می کنید که از دیتابیس CMU [۲۲] استخراج شده‌اند و دارای وضعیت‌های مختلف می‌باشند. که همین باعث سختی در تشخیص آنها در روند شناسایی چهره در الگوریتم‌های هوش مصنوعی می‌باشد. یکی دیگر از مشکلات تشخیص چهره، تصویر پشت صفحه‌ی چهره‌ی شخص می‌باشد. الگوریتم هوش مصنوعی می‌بایست چهره‌ی شخص را در هر حالتی و با وجود شخص در هر زمینه‌ای از تصویر را تشخیص دهد. ممکن است تشخیص چهره با زمینه‌ی سفید بسیار راحت تر به نظر آید تا تشخیص چهره شخص با وجود پشت صفحه‌ای از طبیعت! بنابراین برای کلاسه‌بندی چهره داده‌های مسئله را به دسته‌ی چهره و غیرچهره تفکیک می‌کنیم. در شکل پ.۲ مثالی از تصاویر با پشت زمینه‌های مختلف را ملاحظه می‌کنید.

راه حل‌های مختلفی برای تشخیص چهره در یک تصویر پیچیده وجود دارد، که در اینجا به چند روشِ موجود نگاهی کوتاه می‌شود. باید خاطر نشان کرد که هر کدام از این روش‌ها محتوا و مفهومی کاملاً پیچیده و مفصل دارند و ما فقط در اینجا نگاهی بسیار مختصر به آنها می‌اندازیم.

۱ تشخیص چهره بر اساس یادگیری روی چهره‌ی افراد مختلف و سپس ایجاد یک صفحه‌ی لغزان بر



شکل پ.۱: نمونه‌ای از تصاویر آزمایش که شامل حالت‌های مختلف چهره می‌باشند

روی تصویر مورد آزمایش جهت پیدا کردن تصویری همانند آنچه که آموزش دیده و به عنوان چهره می‌شناسد.

۲ تشخیص چهره بر اساس عوامل هندسی، که بر اساس معیارهای هندسی چهره عمل کرده و تصویر را جهت پیدا کردن آن معیارهای هندسی جستجو می‌کند.

شاید تا به امروز یکی از پرکاربردترین و مؤثرترین راهکار برای شناسایی چهره، استفاده از این روش بوده است. برای بهتر روشن شدن این روش، ابتدا یک سری مفاهیم و مقدمات اولیه‌ای را بیان خواهیم کرد. همانگونه که می‌دانید و در ابتدای هرکاری در زمینه‌ی الگوریتم‌های هوش مصنوعی در تصاویر مفهوم استخراج ویژگی اهمیت پیدا می‌کند که ما در اینجا برای شناسایی چهره به یک سری رهیافت‌های آموزشی احتمالی نیاز داریم. همانطور که می‌دانیم در مرحله‌ی استخراج ویژگی، ویژگی‌هایی که معمولاً از یک تصویر استخراج می‌شوند ممکن است بسیار زیاد باشند و این می‌تواند از لحاظ زمان آموزش بسیار زمان باشد لذا باید یک سری رهیافت‌ها (در اینجا رهیافت‌های بر اساس احتمالات) در نظر گرفته شوند تا تنها ویژگی‌های مفید را تولید کنیم که نتیجه‌ی آن کمتر شدن زمان محاسبات آموزش باشد.
در ادامه تصویری را که به صورت اتفاقی (از یک مجموعه، مثلاً چهره یا غیرچهره) وارد الگوریتم آموزش می‌شود را به عنوان متغییر x در نظر می‌گیریم و باید در نهایت به این نتیجه برسیم که این تصویر احتمالی (که ممکن است تصویری باشد که توسط پنجره‌ی لغزان از تصویر مورد آزمایش استخراج شده است) یا جزء دسته‌ی تصاویر $P(x \mid Face)$ بوده و یا با احتمال $P(x \mid Non\ Face)$ می‌باشد. برای رسیدن به منظور، چندین روش مختلف وجود دارد که از جمله‌ی آنها می‌توان به مدل مخفی مارکوف، شبکه‌های عصبی

شکل پ.۲: نمونه‌ای از تصاویر آزمایش که شامل زمینه‌های مختلف تصویر می‌باشند

و خصوصاً ماشین‌های بردارهای پشتیبان اشاره نمود.

که در این پایان‌نامه هدف ما بر این بوده است که بتوان به کمک ماشین بردار پشتیبان موازی، شناسایی تشخیص چهره در تصاویر را به کمک آن پیاده‌سازی نماییم و سرعت انجام کار و کیفیت انجام کار را با پردازشی که نرم‌افزار MATLAB بر روی تصاویر انجام می‌دهد، مقایسه نماییم.

مراجع

[1] website: *www.wikipedia.com*

[2] C. M. Bishop, *'Neural networks for pattern recognition'*, Oxford University Press, 1995.

[3] C. Burges, *'Tutorial on Support Vector Machines for Pattern Recognition'*, Data Mining and Knowledge Discovery, Vol 2, Issue 2, pp. 121-167,1998.

[4] S. Haykin,*'Neural Networks; A Comprehensive Foundation'*, Second Edition, Pearson Education, Ontarion Canada.

[5] C. Cortes, V. Vapnik, *'Support Vector networks*', Machine Learning, pp. 273–297, 1995.

[6] J. A. Freeman, D. M. Skapura,*'Neural Networks, Algorithm, Applications, and Programming Techniques'*, Addison-Wesley, 1991.

[7] D. Anguita, A. Ghio, S. Pischiutta, S. Ridella,*'A Hardware–friendly Support Vector Machine for Embedded Automotive Applications'*, Proceedings of International Joint Conference on Neural Networks, Orlando, Florida, USA, August 12-17, 2007.

[8] M. R. Latta et all,*'FPGA Implementation of a Support Vector Machine for Classification and Regression'*, WCCI 2010 IEEE World Congress on Computational Intelligence, July, 18-23, 2010 - CCIB, Barcelona, Spain.

[9] Y. Chang, *'PSVM: Parallelizing Support Vector Machines on Distributed Computers'*, Google Research, Beijing, China, 2003.

[10] H. X. Zhao, *'Parallel Support Vector Machines on Multi-core and Multiprocessor Systems'*, The IASTED International Conference Artificial Intelligence and Applications (AIA 2011), Innsbruck, Austria 2011.

[11] B. Heisele, T. Serre, S. Prentice, T. Poggio, *'Hierarchical Classification and Feature Reductio for Fast Face Detection with Support Vector Machines'*, Pattern Recognition, Vol. 36, No. 9, pp.2007-2017, 2003.

[12] H. Rowley, S. Baluja, T. Kanade, *'Neural Network-Based Face Detection'*, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 20, No. 1, pp. 23-38, January, 1998.

[13] S. Romdhani, H. Philip, B. Schlkopf, A. Blake, *'Computationally Efficient Face Detection'*, In Proc. Intl. Conf. on Computer Vision, pp. 695-700, 2001.

[14] P. Viola, M. Jones, *'Rapid Object Detection using a Boosted Cascade of Simple Features'*,In. Proc. Conference on Computer Vision and Pattern Recognition (CVPR01), pp. 511-518 , 2001.

[15] R. Lienhart, J. Maydt, *'An Extended Set of Haar-like Features for Rapid Object Detection'*, IEEE ICIP 2002.

[16] O. P. Ramirez ,*'An FPGA Implementation of Linear Kernel Support Vector Machines'*,ISBN: 1-4244-0690-0/06, IEEE, 2006.

[17] D. Anguita, S. Pischiutta, *'Feed-forward Support Vector Machine Without Multipliers'*, IEEE Transactions On Neural Networks, VOL. 25, 2006.

[18] D. Anguita, A. Boni, *'A Digital Architecture for Support Vector Machines:Theory, Algorithm, and FPGA Implementation'*, IEEE Transactions On Neural Networks, vol. 14, no. 5, September 2003.

[19] Website: *www.ti.com*

[20] Website: *www.Xilinx.com*

[21] C. C. Chang, C. J. Lin, *'LIBSVM: A library for support vector machines'*, ACM Transactions on Intelligent Systems and Technology (TIST), vol. 2, p. 27, 2011.

[22] website: *www.vasc.ri.cmu.edu/idb/html/face/frontal_images*

[23] G. Yang, T. S. Huang,*'Human face detection in a complex background'*, Pattern Recognition, 27(1):53-63, 1994.

[24] A. Lanitis, C. J. Taylor, and T. F. Cootes, *'An automatic face identification system using flexible appearance models'*, Image and Vision Computing, 13(5):393-401, 1995.

[25] T. Leung, M Burl, P. Perona,*'Finding Faces in cluttered scenes using labeled random graph matching'*, In Proc. 5th Int. Conf. on Computer Vision, pages 637-644, MIT, Boston 1995.

[26] Y. Sumi, Y. Ohta, *'Detection of face orientation and facial components using distributed appearance modeling'*, In Proc. Int. Workshop onAuto. Face and Gesture Recogn., pages 254-259, Zurich, 1995.

[27] website: *www.nist.gov/itl/iad/ig/colorferet.cfm*

[28] V. Pavlovic, A. Garg,*'Efficient Detection of Objects and Attributes using Boosting'*, IEEE Conf. Computer Vision and Pattern Recognition, 2001.

[29] C. Bishop, P. Viola, *'Learning and Vision: Descriminative Methods'*, ICCV course on learning and vision, 2003.

[30] X. Shao, D. Sun, *'Development of a New Robot Controller Architecture with FPGA–Based IC Design for Improved High–Speed Performance'*,IEEE Transactions on Industrial Informatics, pp. 312–321, 2007.

[31] D. Anguita,*'Feed–forward Support Vector Machine Without Multipliers'*, DIBE - University of Genoa Via Opera Pia 11A, 16145 Genoa, Italy.

[32] R. Reyna, D. Dragomirescu, D. Houzet,D. Hesteve, *'Implementation of the SVM generalization function on FPGA'*, Proc. of the Int. Signal Processing Conf., Dallas, TX, USA, April 2003.

[33] S. Wong, *'A Sum of Absolute Differences Implementation in FPGA Hardware'*,Delf University of technology, 2008.

[34] X. Shao and D. Sun, *'Development of a New Robot Controller Architecture with FPGA–Based IC Design for Improved High–Speed Performance'*, IEEE Transactions on Industrial Informatics, 3 (2007) 312–321.

[35] J.M. Arnold, *'The Architecture and Development Flow of the S5 Software Configurable Processor'*,The Journal of VLSI Signal Processing, pp. 3–14, 2009.

[36] R. Lienhart and J. Maydt, *'An Extended Set of Haar-like Features for Rapid Object Detection'*, IEEE ICIP 2002.

[37] G. Caffarena, C. Pedreira, C. Carreras, S. Bojanic, and O. Nieto–Taladriz, *'FPGA Acceleration for DNA Sequencing'*,Journal of Circuits, Systems, and Computers, pp. 245–266, 2007.

[38] L. Yip, K. Comanor, J.C. Chen, R.E. Hudson, K. Yao, and L. V andenberghe,*'Array Processing for Target DOA, Localization and Classification based on AML and SVM Algorithms in Sensor Networks'*, 2nd Int. Workshop on Information Processing in Sensor Networks, (2003) 269–284.

[39] D.J. Sebald and J.A. Bucklew, *'Support Vector Machine Techniques for Nonlinear Equalization'*, IEEE Transactions On Signal Processing, 48 (2000) 3217–3226.

[40] X. Nguyen, M.J. Wainwright, and M.I. Jordan, *'Nonparametric Decentralized Detection Using Kernel Methods'*, IEEE Transactions on Signal Processing, 53 (2005) 4053–4066.

[42] G. Ratsch, T. Onoda, K-R. Muller, *'Soft Margins for AdaBoost'*, Machine Learning, 1-35, August 1998.

[43] Duffy and D.P Helmbold, *'Boosting methods for regression'*, Technical report, Department of Computer Science, University of Santa Cruz, 2000.

[44] website: *http://www.boosting.org*

[45] L. Breiman, Bagging predictors, *'Machine Learning'*, pp. 123-140, 1996.

[46] Y. Freund and R.E Schapire. Game theory, *'on-line prediction and boosting'*, In Proc. COLT, pp. 325-332, New York, 1996.

[47] R.E. Schapire, *'A brief introduction to boosting'*, In Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, 1999.

واژه‌نامه انگلیسی به فارسی

نرخ فریم ..... Frame Rate
بهره ..... Gain
تابع گاوسی اساس شعاعی ..... Gassian Radial Basis Function
پردازنده‌های همه منظوره ..... General Purpose Processors
گشتاورهای هندسی ..... Geometric Moments
زبان توصیف سخت افزاری ..... Hardware Descriptin Language HDL
سخت افزار پسند ..... Hardware Freindly
ابر صفحه ..... Hyperplane
پایدار ..... Stably
ورودی/خروجی ..... Input/Output
تابع کرنل ..... Kernel Function
حیله‌ی کرنل ..... Kernel Trick
ضرائب لاگرانژ ..... Lagrange Coefficient
جدول مراجعه ..... Look-Up Table
ماشین بردار پشتیبان چندکلاسه ..... Multi-Class SVM
کلاک ..... Clock
محاسبه‌کننده دیجیتال بر اساس دوران مختصات ..... COrdinate Rotation DIgital Computer
داده کاوی ..... Data Mining
ورود طرح ..... Design Entry
بیت نقلی ..... Carry bit
روند طراحی ..... Design Flow
پردازش سیگنال دیجیتال ..... Digital Signal Processing

پردازنده سیگنال دیجیتال ..... Digital Signal Processor
آرایه منطقی قابل برنامه‌ریزی ..... Field Programmable Gate Array
ممیز ثابت ..... Fixed Point
ممیز شناور ..... Floating Point
واحد کنترل دسترسی میانی ..... Ethernet Medium Access Control (MAC) Unit
ابر صفحه ..... Hyperplane
شبکه‌های عصبی ..... Neural Network
بلادرنگ ..... Real time
مسئله‌ی درجه ۲ ..... Quadratic problem
تک دقتی ..... Single precision
موازنه ..... Trade off
بردارهای پشتیبان ..... Support vector
بر خط ..... On line
خارج از خط ..... Off line
تابع هدف ..... Objective function
مجموعه داده ..... Dataset
مد دورانی ..... Rotation mode
بالارونده‌ی وفقی ..... Adaptive Boosting

Surname: Sahebi Name: Amin

Title: Implementation of Parallel Support Vector Machine (PSVM) on FPGA

Supervisor: Ali Soleimani
Advisor: Alireza Ahmadifard

Degree: Master of Science Subject: Electronic Enginering
Field: Digital Electronic

University of Technology of Shahrood Faculty of Power and Robotic
Date: 2013-2014 Number of pages: 114

Keywords: FPGA;Parallel Support Vector Machine (PSVM); Face Detection, Sum of Absolute (SAD)

Abstract

In this thesis, we present a digital parallel structure for support vector machine (PSVM) feed-forward phase and its implementation on a field programmable gate array. In feed-forward phase implementation respect to fixed-point math imlementation we use hardware friendly kernel to reduce our dependency to multipliers. We analyze our PSVM learning phase in offline mode with MATLAB. our main parallesiem method proposed in SAD computational block.

We use a Spartan3E FPGA for PSVM implementation, in order to reduce usage of multipliers in our algorithm only few amount of resources was used. Its effectiveness is then tested on a face detection problem, where real-time performances are of paramount importance. As a result, our PSVM was simulated by ISE Xilinx and results pepared by ISim Silmulation tool therefor we present the performance of our PSVM according to other references.

University of Technology of Shahrood
Faculty of Power and Robotic

Dissertation Submitted in Partial
Fulfillment of The Requirements For The
Degree of Master of Science in
Electronic Enginering

Implementation of Parallel Support Vector Machine (PSVM) on FPGA

Supervisor

Ali Soleimani

Advisor

Alireza Ahmadifard

by

Amin Sahebi

2013-2014