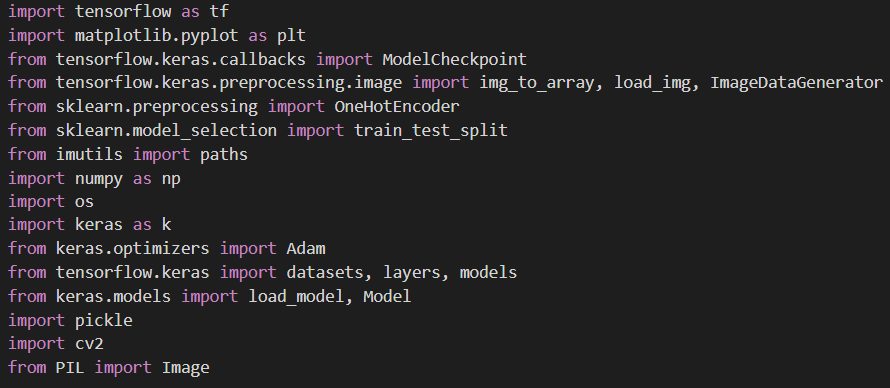
**DOKUMENTASI APLIKASI PENDETEKSI MASKER WAJAH MENGGUNAKAN METODE CNN (Convolutional Neural Network)**

Pada dokumentasi prototype kedua saya ini merupakan aplikasi pendeteksi masker wajah menggunakan metode ***CNN***. Dengan menggunakan dataset yang tersedia di internet, terdiri dari total 3833 file gambar (1915 gambar menggunakan masker, dan *1918* gambar tidak menggunakan masker)

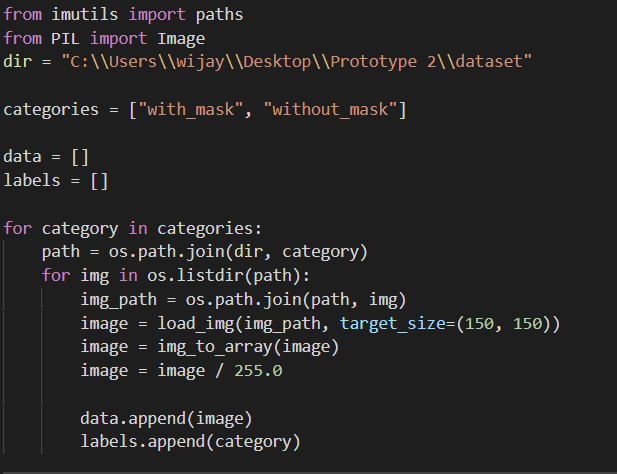
Metode yang saya gunakan untuk memperoleh hasil deteksi masker wajah adalah metode ***CNN*** (Convolutional Neural Network). ***CNN*** (Convolutional Neural Network) merupakan bagian dari deep neural network yang dapat menerima inputan berupa citra (image) dan biasanya digunakan untuk menganalisa citra visual.

Saya juga menggunakan OpenCV (Open Source Computer Vision Library) yang merupakan sebuah library yang digunakan untuk mengolah gambar dan video secara real-time. Computer Vision itu sendiri merupakan kemampuan mesin komputer dalam melihat hingga mampu untuk mengekstrak informasi dari suatu gambar / citra.

**Melakukan import package yang diperlukan**

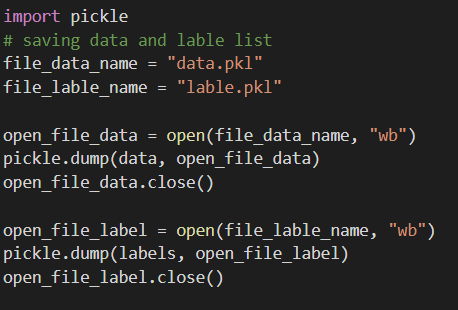


**Menload Data pada folder “Dataset” (data yang berisi gambar orang menggunakan masker dan tidak)**



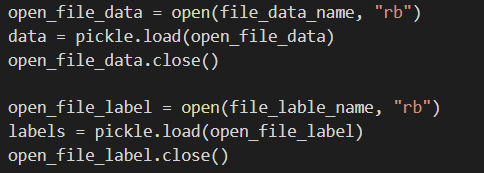
Saya melakukan looping kedalam setiap folder berkategori “with\_mask” dan “without\_mask”, melakukan load image di setiap folder tersebut, menargetkan seluruh gambar berukuran 150 X 150, dan melakukan normalisasi di setiap gambar, kemudian setiap gambar akan di ubah kedalam bentuk array dengan fungsi ***img\_to\_array()*** dan dimasukkan kedalam list ***data*** dan ***labels.***

**Melakukan save pada list *data* dan *label***



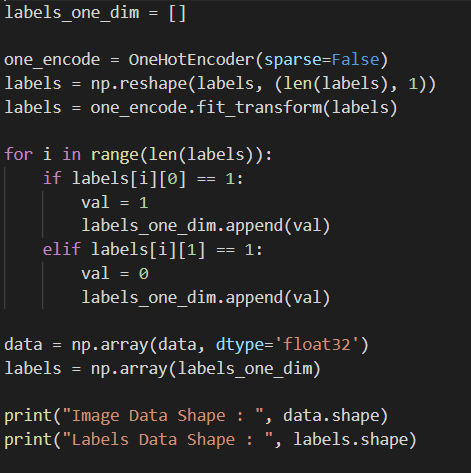
Saya melakukan saving list ***data*** dan ***label*** dalam format ***.pkl***, hal ini saya lakukan agar lebih mudah untuk mengakses data (tidak melakukan looping ke folder terus menerus).

**Membuka file dengan format *.pkl***



Kemudian saya membuka data list yang baru saja saya save sebelumnya menggunakan library ***pickle***.

**Mempersiapkan format data**



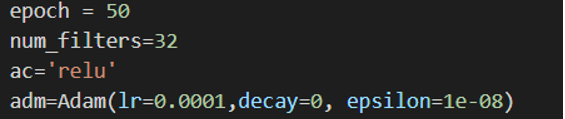


Saya merubah format data ***list*** menjadi ***numpy array,*** dan merubah shape label menjadi array 1 dimensi.

**Melakukan splitting data**

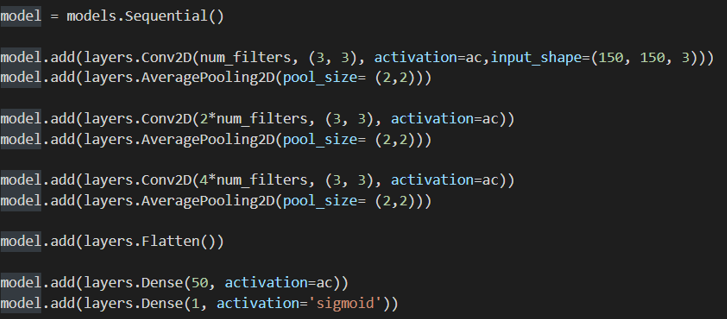
Saya memisahkan data menjadi 80% data training dan 20% untuk data validasi / testing, staritify untuk menyeimbangkan label (0 atau 1) pada saat splitting data berlangsung, dan menyetel ***random\_state*** pada value 42.

Parameter yang digunakan :



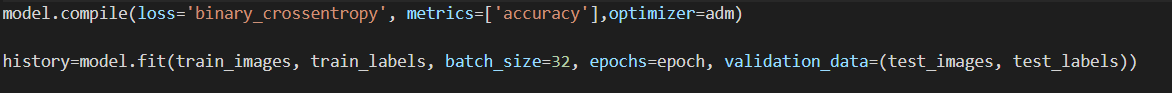
Saya memulai dengan parameter berikut untuk model yang akan saya bangun.

**Model sederhana (Base Model)**

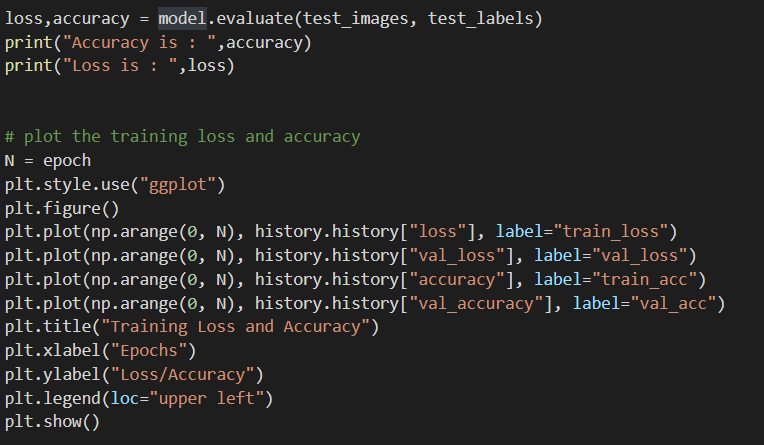


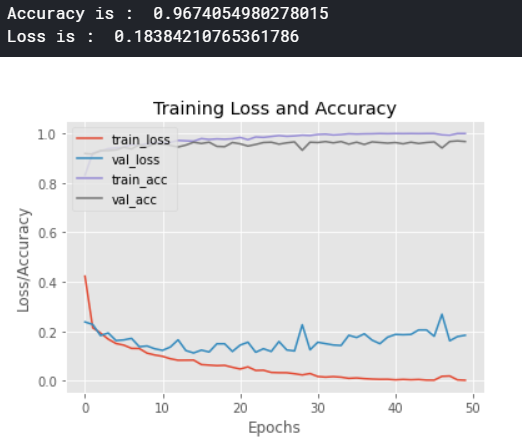
Saya menggunakan arsitektur model seperti di atas untuk memulai training pertama, dengan model sederhana berupa 3 blok layer konvolusi yang diikuti dengan layer average pooling, dan 1 layer dense pada fully connected layer menggunakan ***activation function*** ***sigmoid*** (karena mengharapkan output biner 0 atau 1, maka hanya terdapat 1 output dense).

Kemudian saya melakukan ***model.compile()*** dengan ***loss function*** ‘***binary\_crossentropy***’ karena kita ingin melakukan klasifikasi biner, dan memonitor ***metrics*** akurasi. Setelah itu saya melakukan ***model.fit()*** untuk menjalankan training, dengan ***batch\_size*** sebesar 32 dan epoch sebanyak 50, saya juga melakukan validasi data menggunakan data test yang berupa 20% dari dataset yang dimiliki.



Kemudian saya menampilkan validasi akurasi dan validasi loss pada epoch terakhir, dan menampilkan learning curve (loss dan accuracy pada training maupun validation) menggunakan ***matplotlib.***



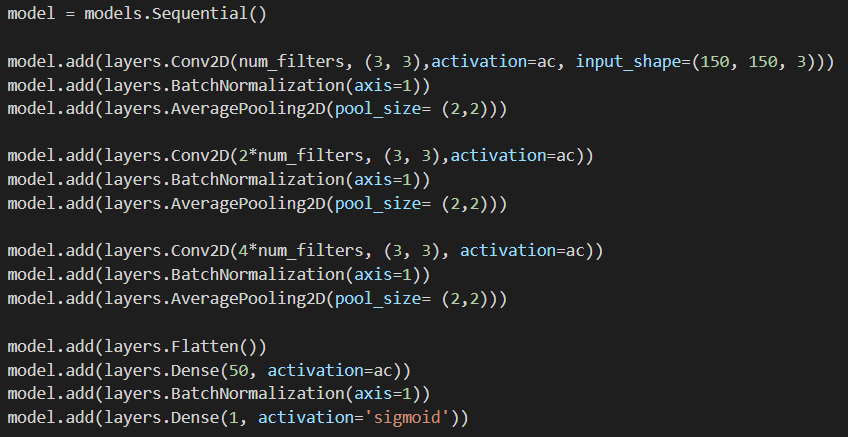


***“Generalization Gap”***

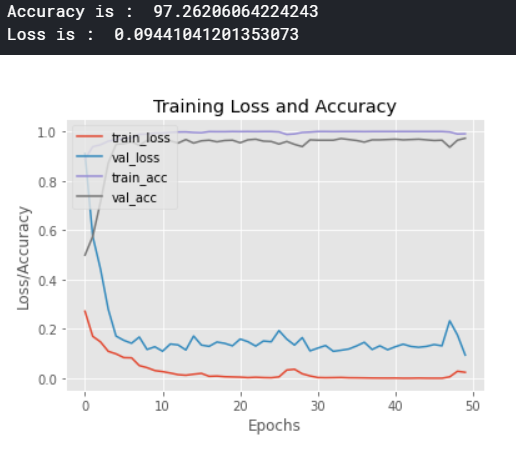
Jika dilihat hasil learning curve diatas, akurasi validasi yang didaptkan cukup bagus di sekitar 96% dan celah di antara akurasi training dan validasi tidak begitu besar (disebut juga ***“Generalization Gap”***), tetapi bisa dilihat bahwa celah di antara loss training dan validasi mulai melebar pada epoch di sekitar 15. Bila saya melanjutkan training loss validasi akan terus meningkat, yang aritnya menandakan model yang saya bangun ini mengalami ***overfitting.***

**Metode regularisasi Batch Normalization**

Saya mencoba menggunakan Teknik ***batch normalization*** untuk lebih menstabilkan learning curve, ini dikarenakan ***batch normalization*** akan melakukan normalisasi di setiap output layer (yang kita terpakan) sebelum output layer tersebut menjadi inputan di layer setelahnya.



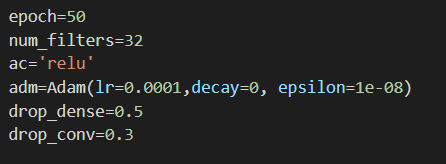
Kemudian saya jalankan dengan ***model.compile()*** dan ***model.fit()*** dengan batch size dan epoch yang sama. Hasilnya seperti berikut :

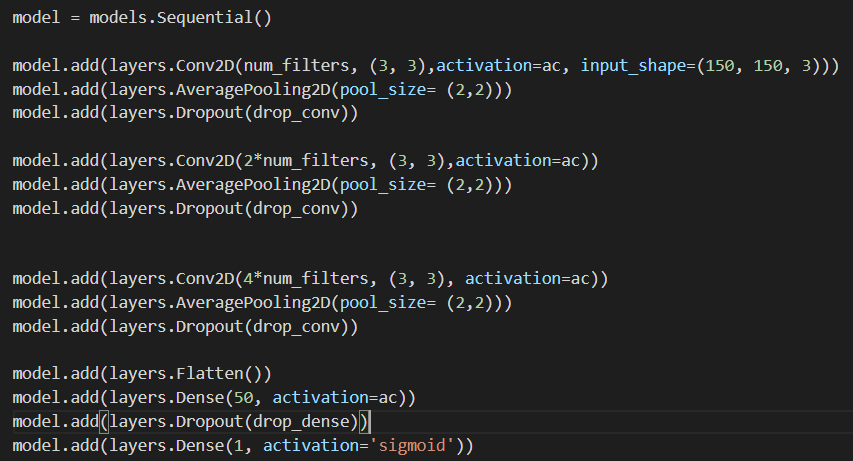


Bisa dilihat bahwa model mengalami peningkatan performa mulai dari akurasi validasi yang cukup tinggi yaitu 97.2% dan loss validasi yang rendah di sekitar 0.0944, tetapi model masih memiliki generalization gap yang cukup besar pada grafik loss.

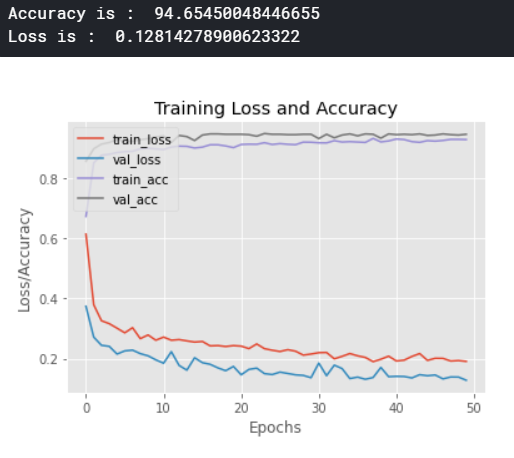
**Metode Dropout**

Saya mencoba metode regularisasi ***Dropout***, yang dimana metode regularisasi ini melakukan pembuangan beberapa neuron secara random dengan dampak mengurangi kapasitas dari model yang dibangun. Dengan adanya metode ini, kasus overfitting pada suatu model akan lebih mudah ditangani.





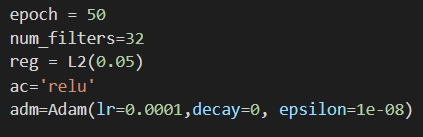
Kemudian hasil yang diperolah akan seperti berikut :



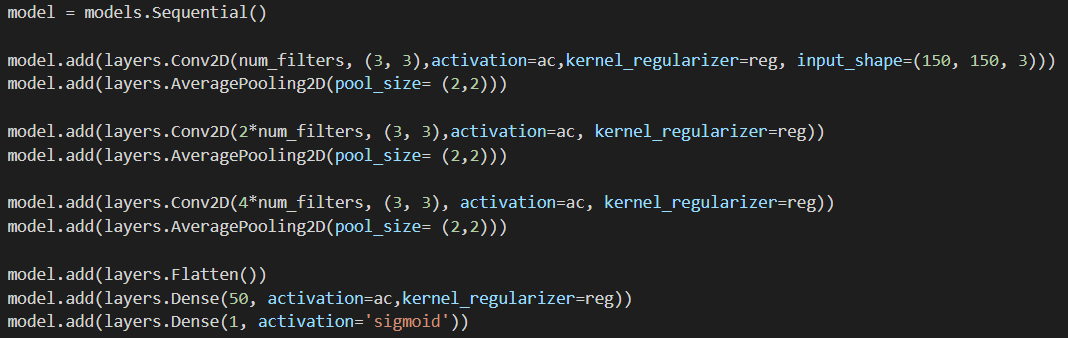
Dengan menggunakan metode ini model mengalami penurunan akurasi validasi di 94.6% dan peningkatan loss validasi disekitar 0.128, tetapi walaupun mengalami penurunan ***generalization gap*** yang terdapat di model-model sebelumnya dapat tertutupi oleh model yang menggunakan teknik ini. Bisa dilihat di ***epoch*** di sekitar 35 sampai 50 ada menyempitan celah terutama diantara akurasi training dan validasi.

**Metode L2 Regularization (Weight Decay)**

Metode ini juga merupakan salah satu metode regularisasi, yang dimana metode ini akan meminimalkan nilai ***weight*** yang besar sehingga stabilitas dari proses pembelajaran akan lebih terjaga. Dengan menggunakan ***import L2*** yang disediakan oleh ***keras.regularizers,*** berikut merupakan parameter yang digunakan :

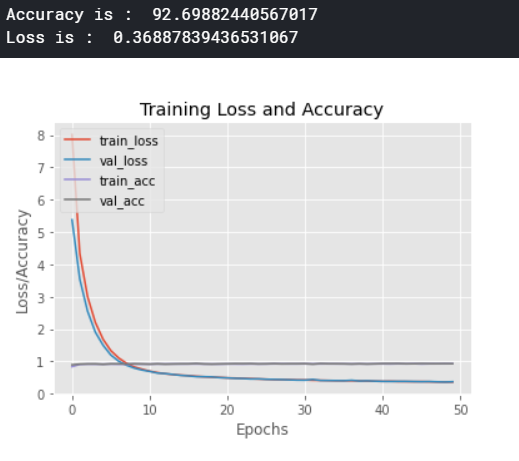


Model yang digunakan :



Dengan menambahkan parameter ***kernel\_regularizer*** untuk pengimplementasian metode L2 regularization (***Weight Decay***).

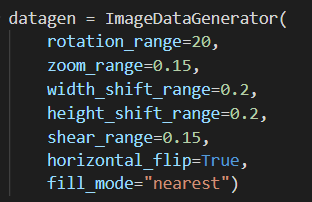
Setelah itu saya menggunakan ***model.compile()*** dan ***model.fit()*** untuk menjalankan training dengan parameter-parameter yang sama seperti sebelumnya, dan menghasilkan hasil seperti berikut :



Dengan menggunakan metode ***L2 regularization*** loss pada validasi dan training dimulai dengan sangat tinggi (sekitar 5.3 dan 8.0), begitu pula dengan akurasi validasi dan training. Performa akurasi dan loss yang di dapatkan pada metode ini sedikit lebih kurang dibandingkan metode regularisasi yang di coba sebelumnya, tetapi dengan ***generalization gap*** yang sangat kecil (hampir tidak terlihat) dengan learning curve yang sangat stabil.

**Data Augmentation**

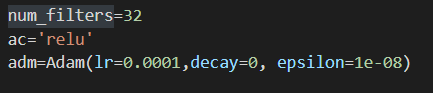
Kemudian saya mencoba untuk melakukan teknik data augmentation untuk mendapatkan contoh gambar yang lebih bervariasi, dengan tujuan untuk melatih model agar lebih general dalam memprediksi. Dengan menggunakan ***ImageDataGenerator*** saya melakukan augmentasi data sebagai berikut :



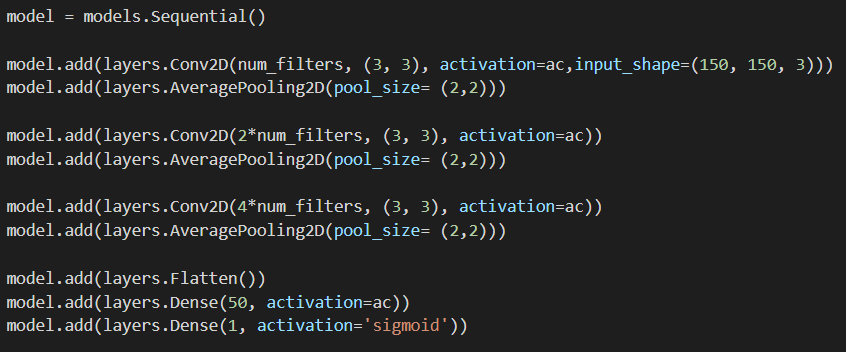
Kemudian saya melakukan augmentasi data sebanyak 32 data / gambar per batchnya, seperti gambar berikut :

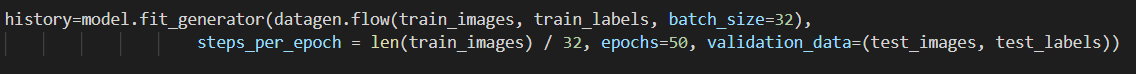


Dengan parameter seperti berikut :

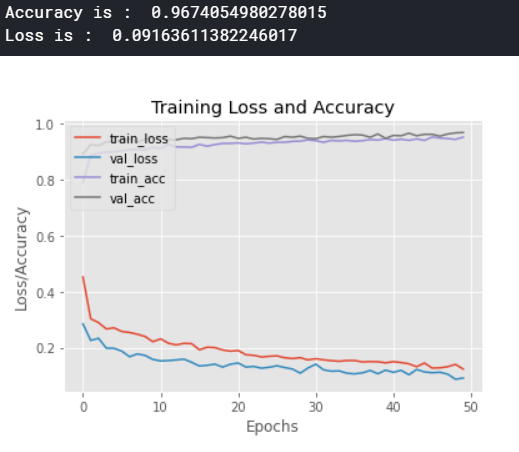


Kemudian dengan model seperti berikut :



Kemudian saya melakukan ***model.compile()*** seperti sebelumnya, tetapi dengan ***model.fit\_generator()*** karena saya menggunakan ***ImagerDataGenerator()*** untuk pengimplementasian data augmentasinya. Dengan ***epoch*** sebanyak 50 dan ***step\_per\_epoch*** sebesar banyak data training dibagi dengan 32 ***(3066 / 32 = 96 batch)***

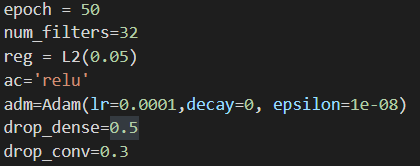
Hasil yang di dapatkan sebagai berikut :



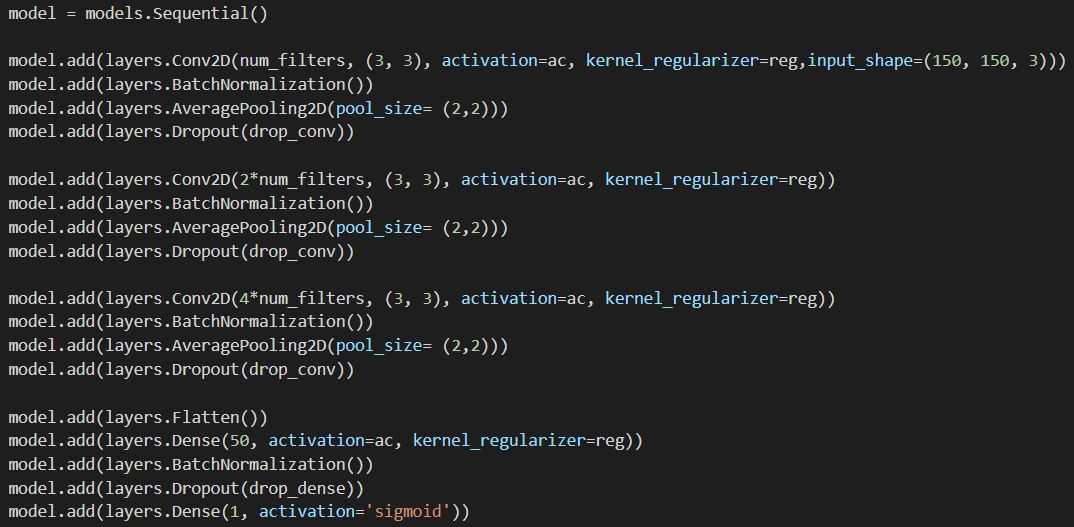
Dari hasil di atas, model menggunakan metode augmentasi data menghasilkan performa learning curve yang cukup baik, dengan mendapatkan akurasi validasi di atas 95% dan validasi loss dibawah 0.1 . Akurasi terus mengalami kenaikan dari ***epoch*** pertama hingga ***epoch*** terakhir, begitu juga dengan loss yang terus mengalami penurunan yang dimana hal ini memberikan kesimpulan bahwa ***overfitting*** sudah berkurang dengna menggunakan metode ini.

**Metode Data augmentation + Dropout + BatchNormalization + L2 Regularization (Weight Decay)**

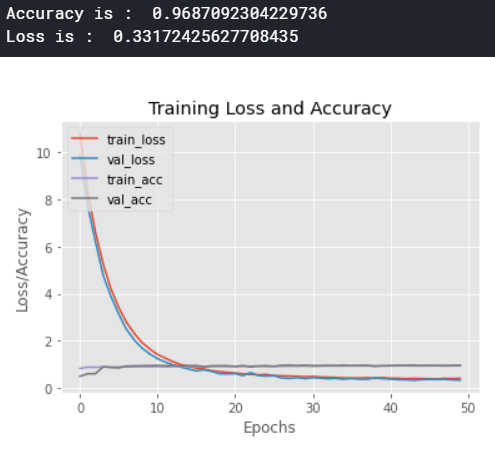
Saya mencoba untuk menggabungkan teknik data augmentasi dan ketiga metode regularisasi yang sudah saya coba sebelumnya, dengan parameter seperti berikut :



Dengan model seperti berikut :



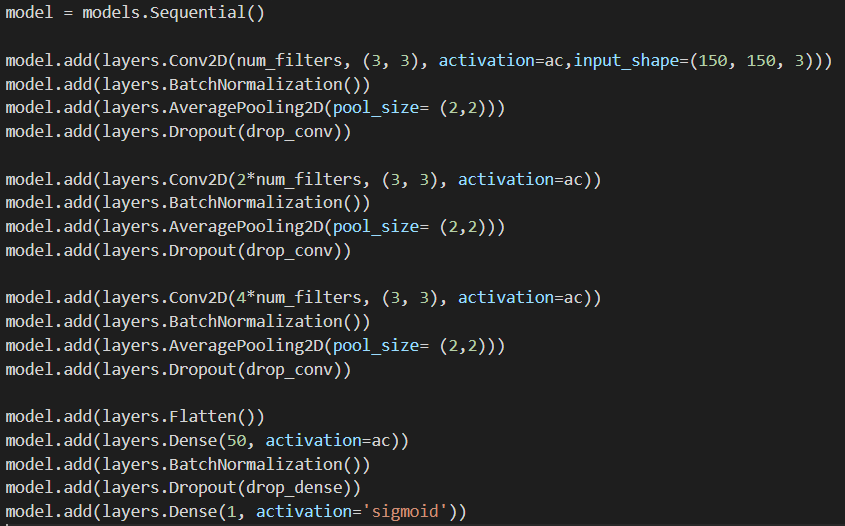
Kemudian hasil yang di dapatkan seperti berikut :



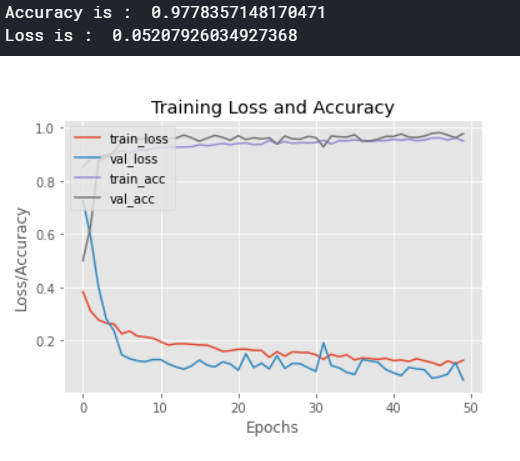
Dengan menggunakan metode ini, bisa dilihat loss pada training dan validasi di mulai dengan nilai yang sangat tinggi (disekitar 11), dan kemudian mulai turun ke nilai yang stabil pada epoch ke 15. Pada metode ini model mendapatkan performa akurasi validasi di 96.8% dan loss validasi 0.331, dengan learning curve yang sangat stabil.

**Metode Data Augmentation + Dropout + Batch Normalization**

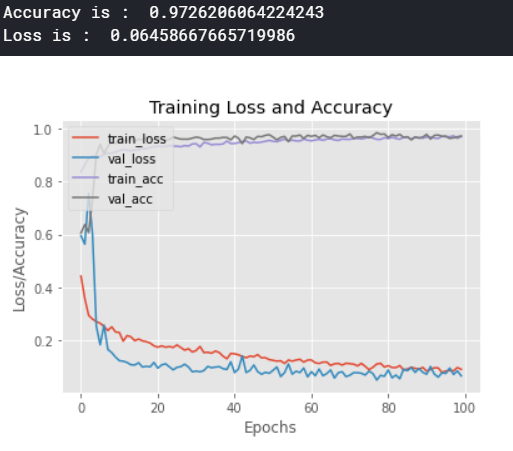
Pada metode kali ini, saya mencoba untuk menghilangkan metode regularisasi L2. Dikarenakan menurut penelitian yang di publish oleh ***Twan Van Laarhoven*** dari universitas ***Cornell University*** bahwa metode regularisasi L2 tidak akan bekerja bila di kombinasikan oleh metode ***Normalisasi***. Maka dari itu model yang saya gunakan akan seperti berikut :



Hal yang saya lakukan hanya menghilangkan parameter ***kernel\_regularizer.*** Kemudian saya masih menggunakan parameter ***model.compile()*** yang sama dan ***model.fit\_generator()*** yang sama. Hasil yang di dapatkan seperti berikut :

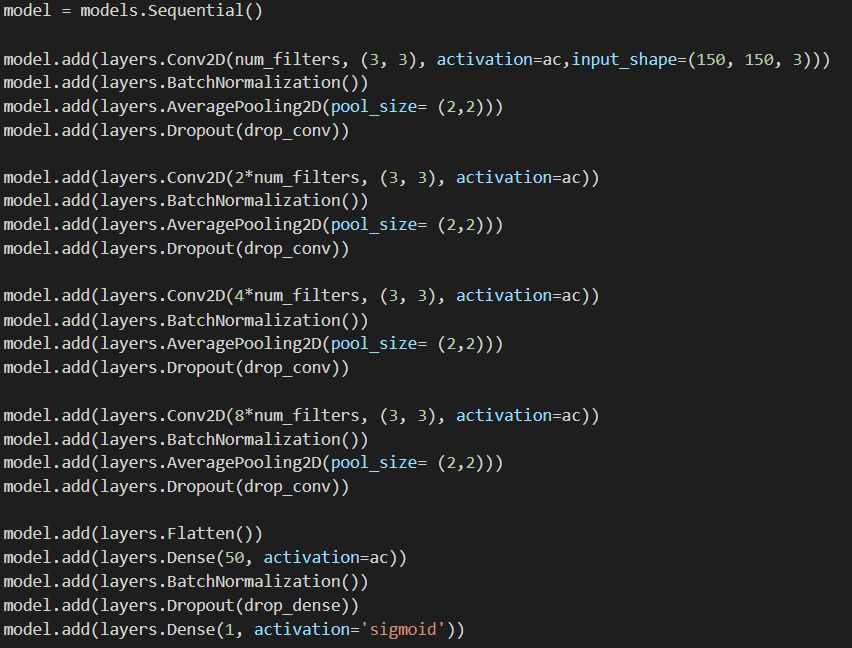


Akurasi dan loss validasi yang dihasilkan oleh model pada metode ini cukup baik yaitu di sekitar 97.7% dan 0.052. Tetapi bisa dilihat “***generalization gap***” pada loss masih terlihat cukup lebar, maka dari itu saya mencoba untuk melakukan training dengan epoch lebih, yaitu 100 epoch. Hasil yang di dapatkan seperti berikut :

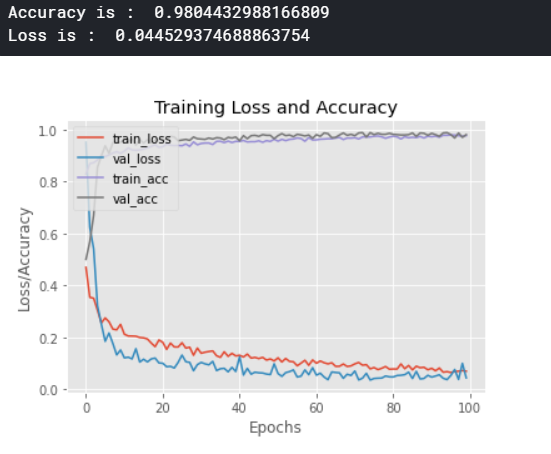


Hasil dari akurasi dan loss yang di hasilkan cukup mirip dengan model dengan 50 ***epoch*** (loss validasi yang meningkat sekitar 0.01), tetapi bisa dilihat ***“generalization gap”*** mulai merapat pada epoch di sekitar 85.

Kemudian saya mencoba untuk menambah 1 blok layer konvolusi dengan tujuan untuk meningkatkan performa dari model. Model yang digunakan sebagai berikut :

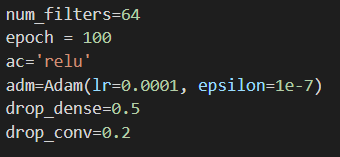


Dengan menambahkan satu blok layer konvolusi, model menghasilkan hasil seperti berikut :

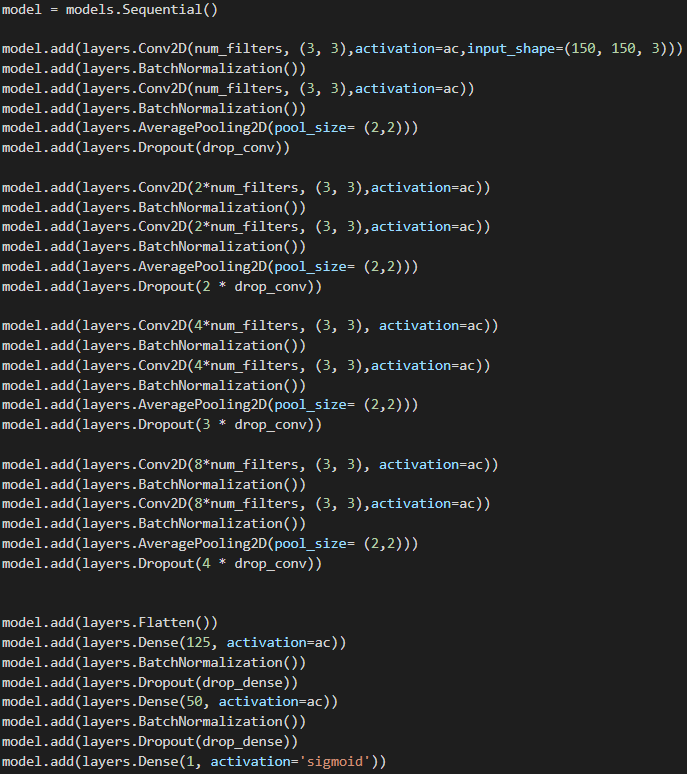


Model mengalami sedikit peningkatan akurasi validasi menjadi 98% dan penurunan loss validasi menjadi 0.044, dengan learning curve dari loss dan akurasi terus mengalami penurunan, yang dimana artinya model ini tidak mengalami overfitting.

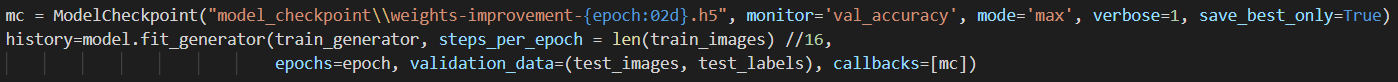
Kemudian saya mencoba untuk melakukan beberapa perubahan lagi dengan tujuan untuk meningkatkan performa dari model yang dibangun sebelumnya. Saya merubah sedikit parameter yang saya gunakan sebelumnya menjadi seperti berikut :



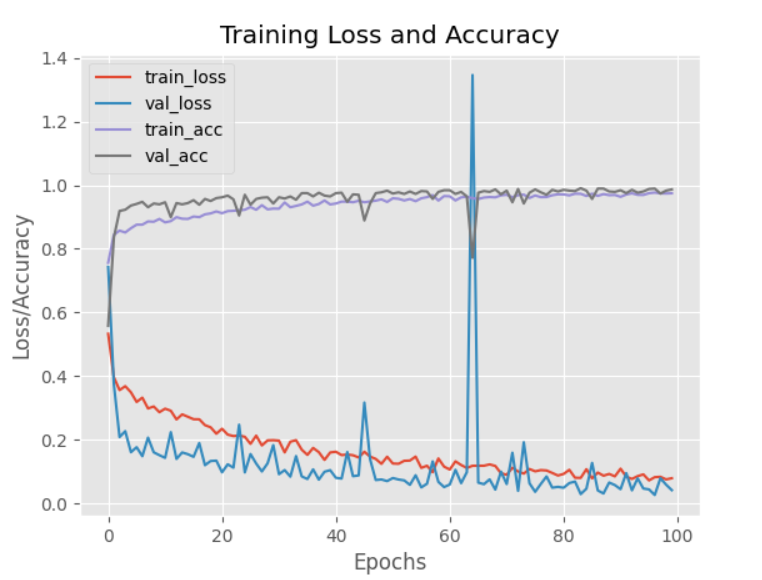
Saya memulai dengan 64 filter konvolusi (sebelumnya saya memulai dengan 32 filter), menambahkan ***epoch*** menjadi 100, merubah parameter ***epsilon*** menjadi 1e-7 (yang dimana nilainya lebih besar dari nilai sebelumnya, yaitu 1e-8), dan menurunkan base dropout konvolusi menjadi 0.2 atau 20%. Kemudian model yang saya gunakan seperti berikut :



Saya menambahkan 1 layer konvolusi ekstra di setiap blok (total 2 layer konvolusi di setiap blok), dan menambahkan 1 layer dense di fully connected layer (total 2 layer dense) dengan tujuan untuk menambahkan kompleksitas daripada model itu sendiri. Disetiap layer konvolusi akan saya terapkan ***Batch Normalization*** dan ***Dropout*** yang terus meningkat di setiap akhir blok, begitu pula pada setiap layer dense pada fully connected layers.

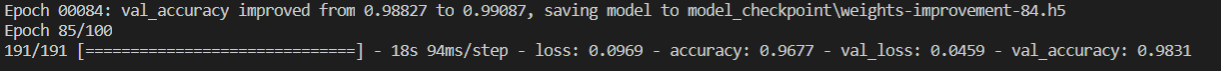
Kemudian saya juga menambahkan metode callback berupa ***ModelCheckpoint***, untuk melakukan save pada performa model terbaik di setiap berjalannya epoch ke -n. 

saya melakukan monitor pada ***‘val\_accuracy’*** (validasi akurasi) maksimal pada setiap epoch dan melakukan save performa model terbaik dalam 100 epoch yang saya jalankan kedalam folder ***“model\_checkpoint”*** dengan format file ***“weight-improvement-{epoch ke-n}”*** dengan format ***.h5***. Kemudian menambahkan parameter ***“callbacks”*** pada ***model.fit\_generator()*** untuk mengimplementasikan ***modelcheckpoint*** yang sudah dibuat. Saya juga merubah step\_per\_epoch (batch Size) menjadi lebih kecil, yaitu 16 ***(3066 / 16 = 191 batch)*** sehingga batch menjadi semakin banyak per epochnya. Hasil dari learning curve yang dihasilkan seperti berikut :



Bisa dilihat model menghasilkan performa yang baik, yang dimana akurasi terus naik dan loss yang terus turun dengan berjalnnya epoch. Walaupun ada spiking (kenaikan yang sangat tinggi secara mendadak) di sekitar epoch 62an, tetapi setelah itu loss Kembali ke posisi semula dan terus menurun.

Kemudian pada model mengalami performa dengan akurasi validasi terbaik (99%) pada epoch ke 84, setelah itu akurasi validasi tidak mengalami peningkatan dari epoch ke 85 sampai ke 100.

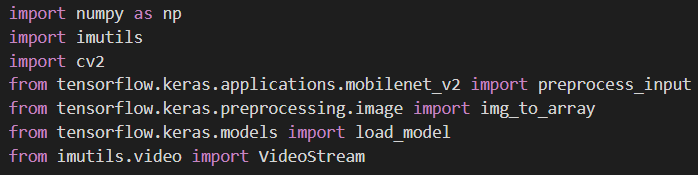




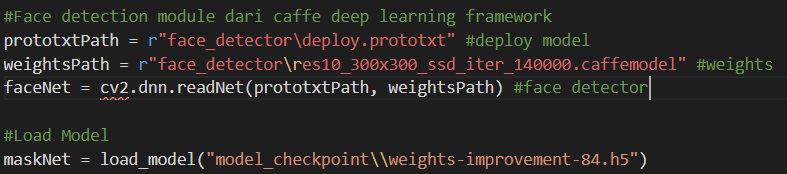
Kemudian model yang memiliki performa terbaik tersebut disimpan dalam folder ***model\_checkpoint*** dengan nama file ***weights-improvement-84.h5***.

**Mendeteksi menggunakan OpenCV**

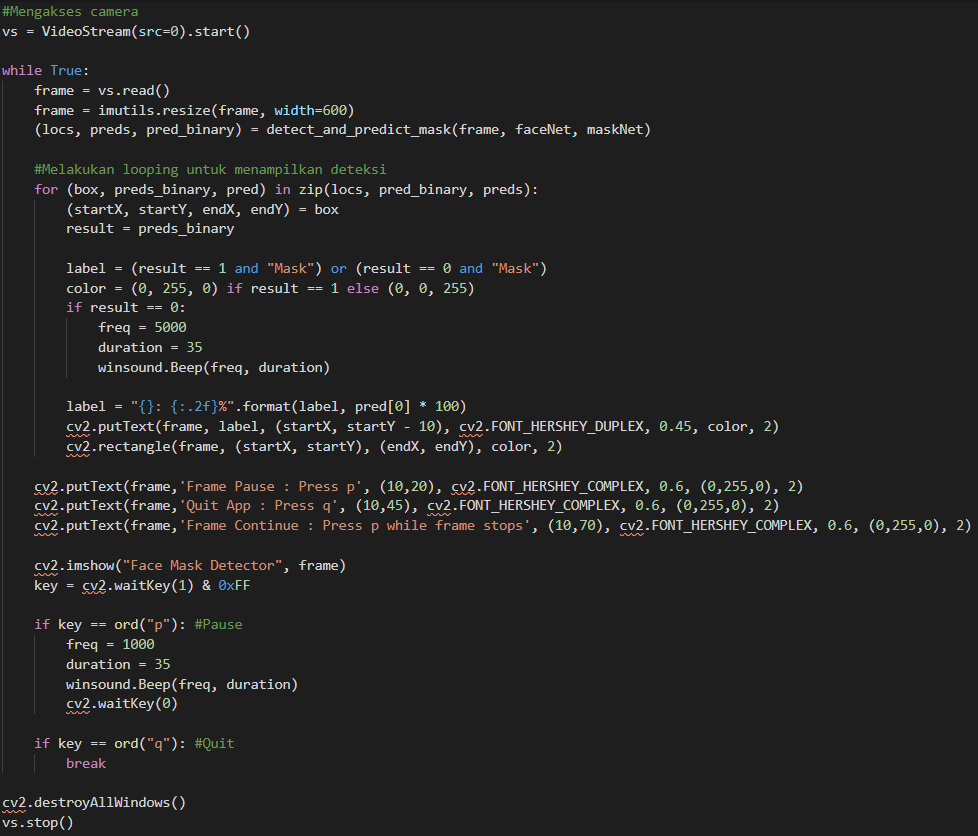
Saya melakukan import beberpa library yang diperlukan terlebih dahulu



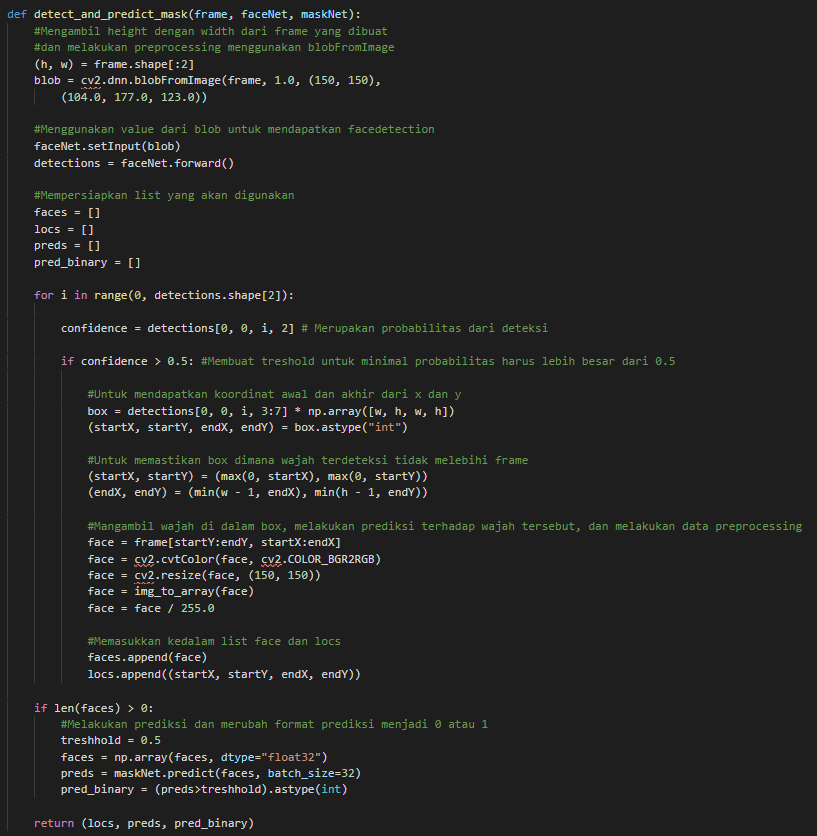
Kemudian menload model yang akan digunakan dan face detector yang sudah pernah dibuat sebelumnya dengan format ***.prototxt*** dan ***.caffemodel.***

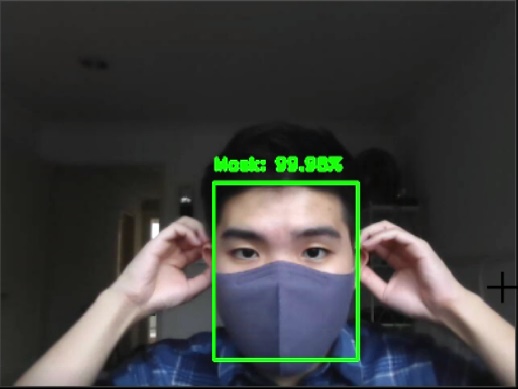


Kemudian menjalankan loop supaya video yang di tangkap perframe dapat melakukan deteksi terus menerus.



Terdapat fungsi untuk melakukan prediksi, yang menerima parameter berupa frame, face detector, dan model yang di bangun.



Pada fungsi ini pada dasarnya melakukan ekstraksi wajah yang sudah di predict melalui model yang di dapatkan melalui ***faceNet***, kemudian ekstraksi tersebut kita gunakan Kembali untuk mendeteksi apakah wajah tersebut menggunakan masker atau tidak. Berikut beberapa cuplikan dari aplikasi yang di bangun :

