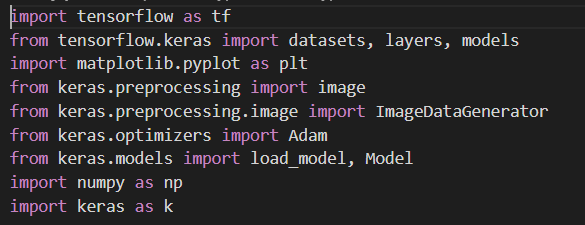
**DOKUMENTASI KLASIFIKASI GAMBAR MENGGUNAKAN MODEL CNN DENGAN DATASET CIFAR-10**

Pada dokumentasi prototype pertama saya ini merupakan klasifikasi gambar menggunakan dataset ***CIFAR-10*** dan model ***CNN*** (Convolutional Nerual Network). Pada dokumentasi kali ini saya akan menjelaskan tahapan pengembangan model yang saya kembangkan.

***CIFAR – 10*** (Canadian Institute For Advance Research - 10) merupakan dataset yang berisi kumpulan gambar yang biasa digunakan untuk melatih model *machine learning* dan *computer vision.* ***CIFAR-10***memiliki total 60.000 gambar berwarna (***RGB***) berukuran 32 x 32 pixel dengan 10 kelas berbeda, 10 kelas itu diantara lain adalah pesawat terbang, mobil, burung, kucing, rusa, anjing, katak, kuda, kapal, dan truk. Masing -masing kelas memiliki 6000 gambar.

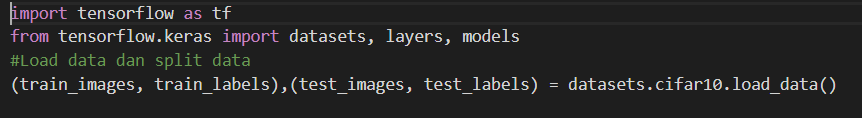
***CNN*** (Convolutional Neural Network) merupakan salah satu metode *machine learning* yang merupakan pengembangan ***Multi Layer Perceptron*** yang di desain untuk mengolah data dua dimensi, seperti gambar / citra.

**Melakukan import package yang diperlukan**

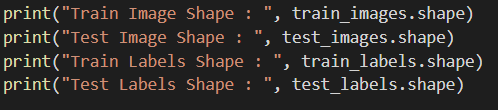


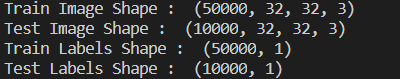
**Menload Dataset CIFAR-10**

pada langkah awal saya menload data CIFAR-10 melalui dataset API dari tensorflow di dalam keras dan melakukan pembagian data, seperti :



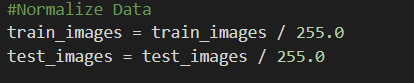
Hasil dari pembagian data :





**Normalisasi Data**

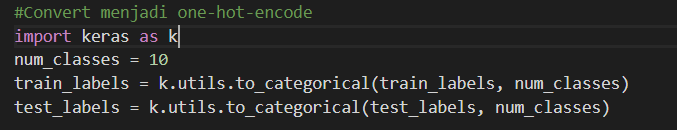
Kemudian saya melakukan normalisasi data pada training dan testing image, agar setiap nilai dari pixel pada gambar memiliki rentangan nilai dari 0 sampai dengan 1. Tujuan melakukan normalisasi adalah untuk meringankan dan mempercepat komputasi yang akan dilakukan pada model nanti.



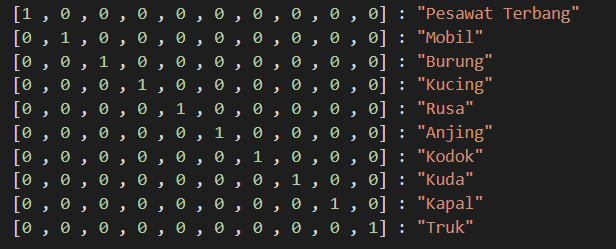
Mengapa dibagi dengan nilai 255 ? karena rentangan maksimal pada channel RGB adalah 255, maka dari itu nilai tidak akan melebihi dari 1.

**Melakukan konversi label menjadi one-hot-encode**

Pada dataset CIFAR-10 label yang berupa pesawat terbang, mobil, burung, kucing, rusa, anjing, katak, kuda, kapal, dan truk merupakan nilai 0 sampai dengan 9 secara berurut, seperti contoh nilai 0 merupakan pesawat terbang, 1 merupakan mobil, dan seterusnya. Dengan menggunakan fungsi *to\_categorical()* pada *keras.utils*.

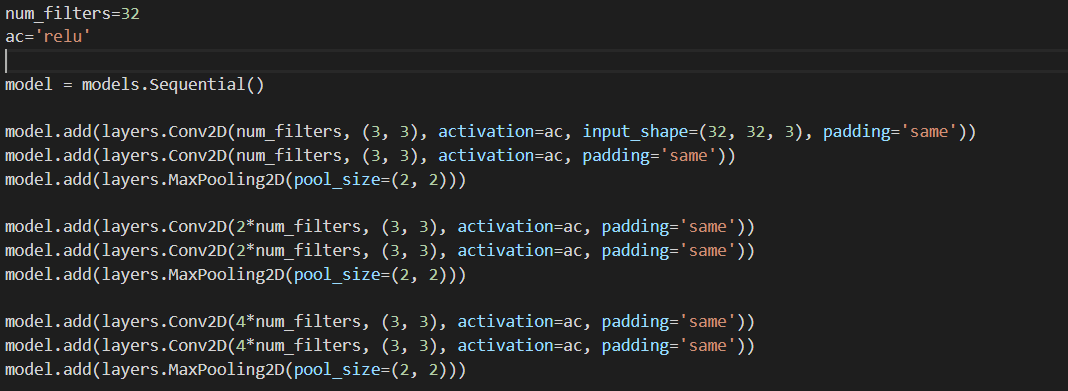


One – hot – encoding merupakan proses untuk merubah bentuk nilai kategorikal menjadi bentuk yang bisa digunakan algoritma *machine learning* untuk melakukan prediksi lebih mudah. Contoh one – hot – encoding :



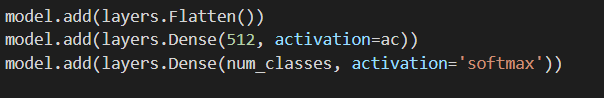
**Model CNN**

Model yang saya gunakan untuk model dasar pada prototype ini merupakan arsitektur model VGG, yang dimana arsitektur ini menggunakan lapisan konvolusi dengan filter 3x3 dan diikuti dengan lapisan max pooling, saya akan menggunakan 3 blok arsitektur VGG untuk dijadikan model awal.

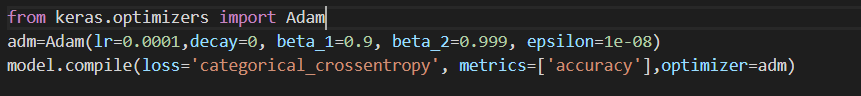


Saya menggunakan sequential class yang disediakan oleh *Keras* *API,* pada lapisan ini berfungsi untuk melakukan ekstraksi fitur pada image yang sedang kita training.

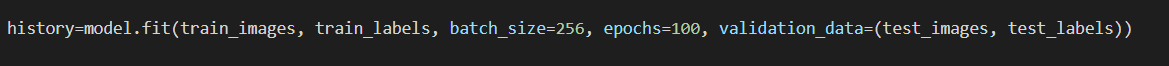
Kemudian setelah melalui proses ekstraksi fitur, akan dilanjutkan ke FC layer (Fully Connected Layer). Dimuali dengan melakukan Flatten pada fitur – fitur yang baru saja di ekstraksi, kemudian disambungkan ke FC layer dengan 10 output layer dengan menggunakan *softmax activation function.*



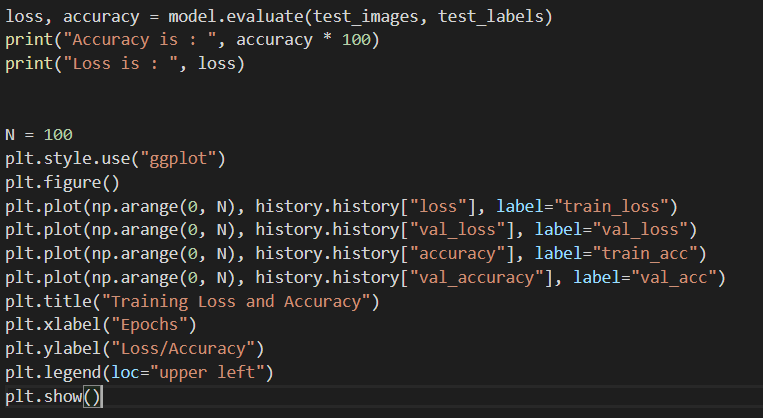
Kemudian melakukan compile model dengan class *compile* yang disediakan oleh *keras API,* memulai dengan Adam optimizer dengan leraning rate = **0.0001**,beta\_1 = **0.9**, beta\_2 = **0.999**, epsilon = **1e-08**. Model ini akan melakukan optimisasi dengan ***categorical\_crossentropy loss function*** yang dibutuhkan untuk melakukan klasifikasi multi kelas dan akan terus memonitor akurasi daripada klasifikasi.



Kemudian melakukan training dengan *model.fit()*, dengan ***batch\_size = 256*** dan ***100 epoch***, dengan melakukan validasi data menggunakan data testing (10000 data)



Melakukan pengecekan akurasi dan loss pada validation, dan melakukan ploting akurasi dan loss



Ini merupakan hasilnya (hasil sama dengan akurasi dan loss validasi pada epoch ke - 100, karena saya melakukan ***model.evaluate()*** pada data yang sudah saya gunakan untuk ***validation\_data()***)

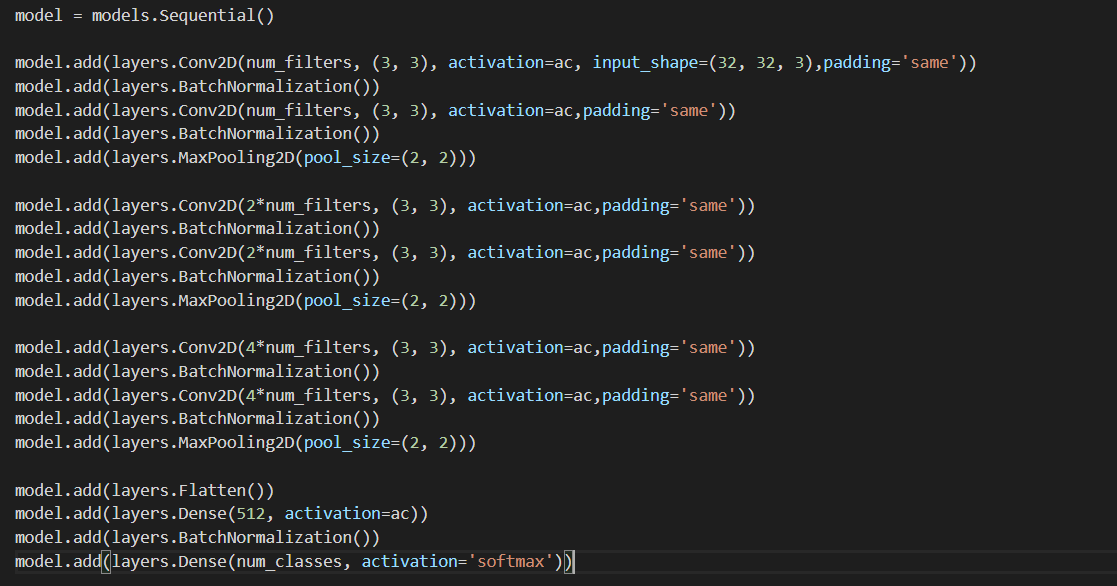


Bisa dilihat bahwa model kita mengalami overfitting mulai dari epoch ke – 15, di tandai dengan turunnya validasi akurasi dan naiknya validasi loss .

**Menggunnakan metode regularisasi**

**Batch Normalization**

Untuk menangani hal tersebut, saya mencoba untuk melakukan ***BatchNormalization***, ***BatchNormalization*** merupakan merupakan Teknik didmana melakukan standarisasi setiap input layer pada setiap mini-batch, ***BatchNormalization*** berperan untuk regularisasi untuk mencegah overfitting.



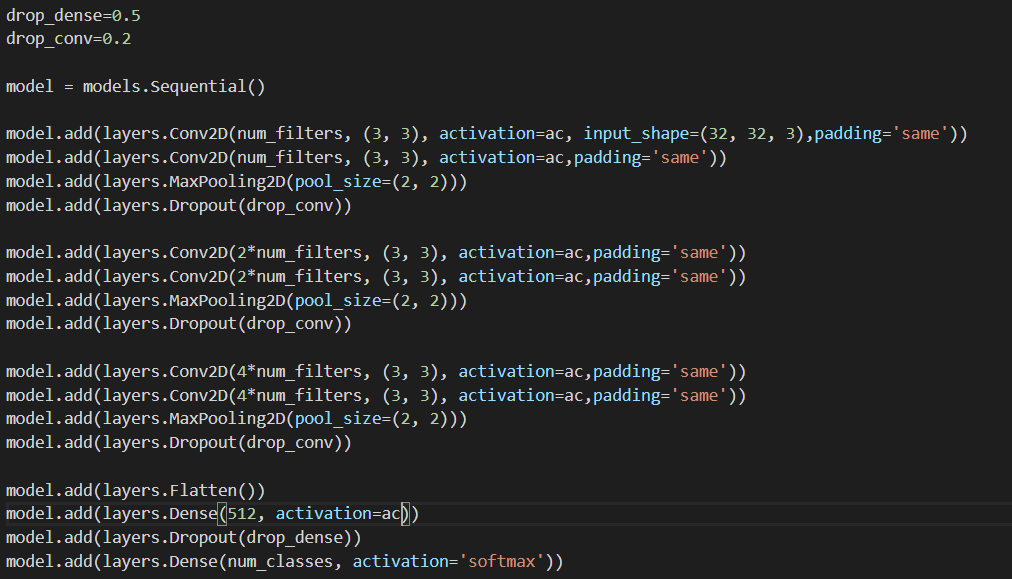
Kemudian melakukan ***model.compile()***, ***model.fit()***, dan menampilkan plot akurasi dan loss, dan menghasilkan hasil seperti berikut :



Bisa dilihat bahwa dengan melakukan ***BatchNormalization*** tidak begitu efektif pada model dasar yang saya gunakan, model masih mengalami overfitting tetapi validasi loss mengalami penurunan dan akurasi validasi mengalami penurunan juga.

**Dropout**

Kemudian saya mencoba untuk menggunakan Dropout layer untuk melakukan regularisasi, Dropout merupakan teknik yang secara acak akan melakukan “drop” pada neurons yang ada di jaringan neural network (sesuai dengan persentasi dropout yang di berikan). Dengan adanya dropout, model neural network akan sedikit mengalami penurunan performa tetapi model dapat lebih menyamaratakan (generalize) data yang belum pernah model tersebut lihat.



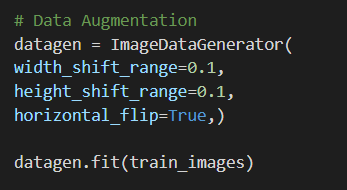
Melakukan ***model.compile()***, ***model.fit()***, dan menampilkan plot akurasi dan loss, dan mengahasilkan hasil seperti berikut :



Bisa dilihat bahwa metode regularisasi menggunakan dropout lebih efektif disbanding menggunakan BatchNormalization(), dengan adanya peningkatan validasi akurasi menjadi 79.49% dan penurunan validasi loss menjadi 0.642.

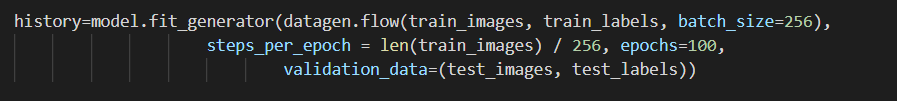
**Data Augmentation**

Setelah mencoba metode regularisasi seperti yang sudah dilakukan sebelumnya, saya memilih metode dropout untuk menjadi metode regularisasi. Dengan menerapkan metode dropout, model masih mengalami sedikit overfitting di sekitar epoch ke – 40 dengan adanya penurunan akurasi validasi dan kenaikan loss validasi. Maka dari itu saya mencoba metode data augmentation untuk menambahkan efek regularisasi. ***Data Augmentation*** pada dasarnya akan melakukan duplikasi pada dataset training dengan menabahkan modifikasi pada gambar yang di duplikasi.



Saya melakukan modifikasi berupa perubahan lebar dan tinggi dari gambar sebanyak 10% dan melakukan perputaran horizontal pada gambar, dengan menggunakan ***ImageDataGenerator*** ***class*** dari keras.

Kemudian melakukan training menggunakan ***model.fit\_generator()***, dengan parameter pertamanya merupakan iterasi dari setiap gambar per batchnya (***datagen.flow()***).



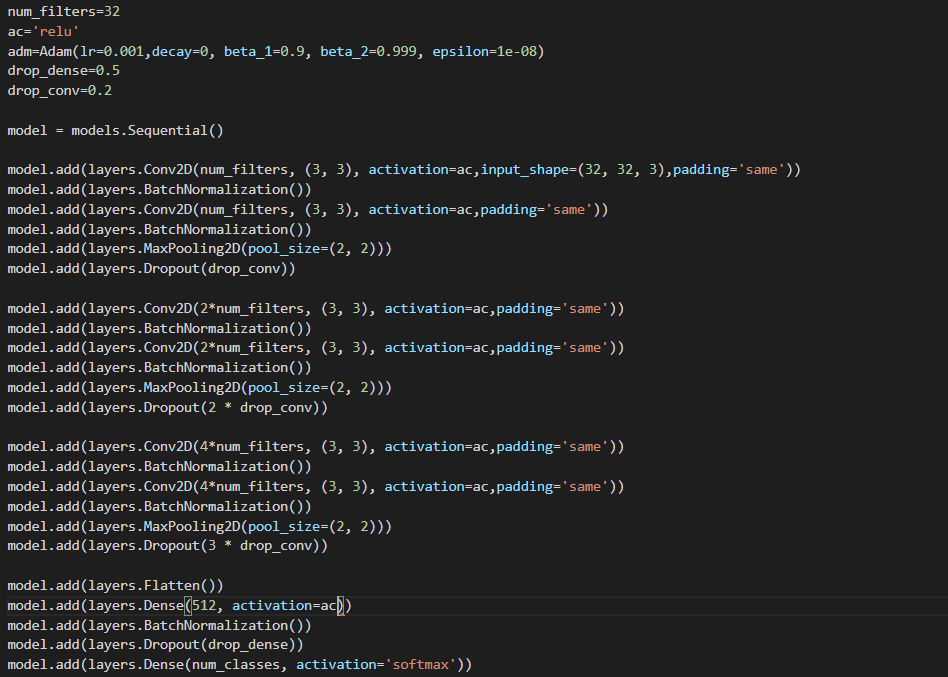
Metode data augmentation di jalankan dengan model dasar (tanpa ***dropout*** dan ***batchnormalization***), dengan optimizer yang sama (Adam Optimizer). Hasil yang didapatkan seperti dibawah ini :



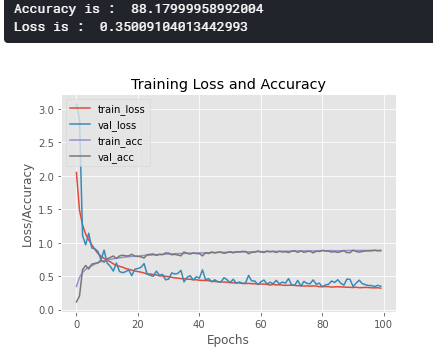
Validasi akurasi yang dihasilkan dengan metode data agumentation yaitu 80.70% dan validasi loss 0.649, jika di bandingkan dengan performa model dasar (tanpa ***dropout*** dan ***batch normalization***) yang tidak menggunakan metode augmentation, akurasi meningkat dari 72% menjadi 80%, dan loss yang berkurang dari 2.293 menjadi 0.649.

**Data Augmentation + Dropout + Batch Normalization**

Pada tahap ini saya melakukan kombinasi antara metode ***data augmentation***, ***Batch Normalization***, dan ***dropout regularization***. Saya melakukan sedikit modifikasi pada dropout yang akan digunakan, yaitu dengan meningkatkan persentase dropout di blok ke – 3 (20 % persen di blok pertama, 40% di blok ke dua, dan 60% di blok ke tiga) dan 50% di bagian fully connected layer.

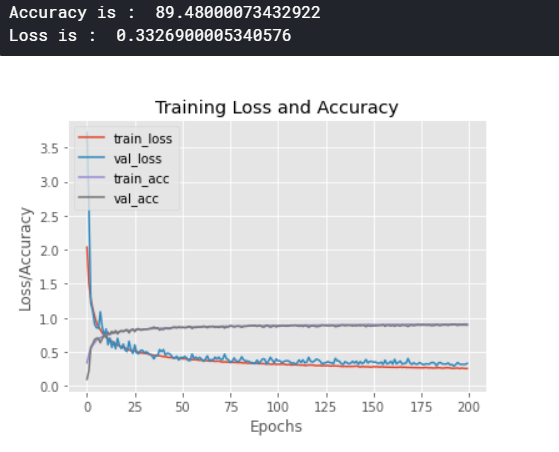


Kemudian saya melakukan ***model.compile()*** dengan optimizer yang sama (Adam) dan menjalankan training dengan ***model.fit\_generator()***, kemudian mendapatkan hasil seperti ini :



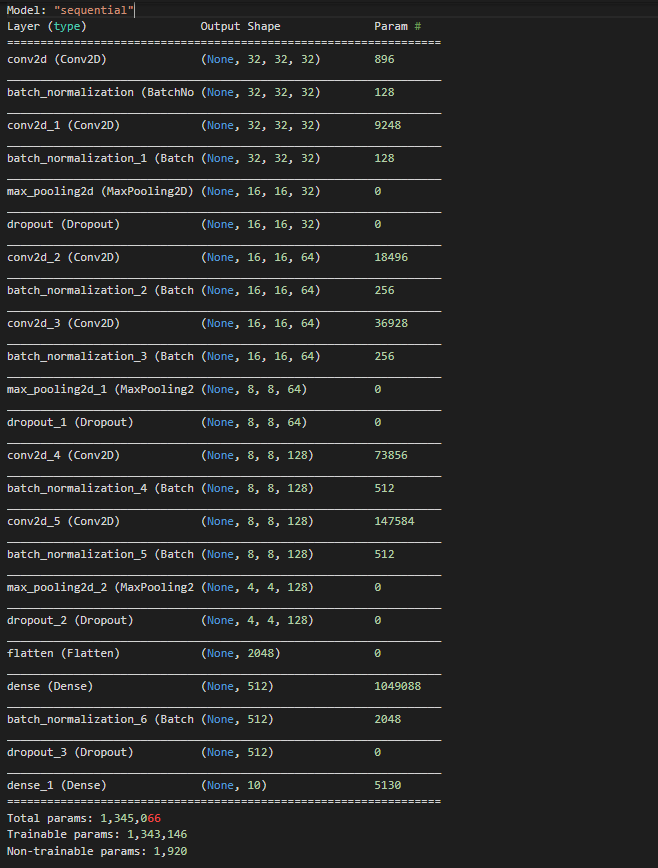
Bisa dilihat performa model mengalami kenaikan validasi akurasi dari 80.70% menjadi 88.17% dan validasi loss yang berkurang dari 0.649 menjadi 0.350. Overfitting yang terjadi pada model sebelumnya (training hanya menggunakan data augmentation) sudah cukup di tutupi oleh performa model ini, bisa dilihat dari nilai akurasi validasi dan loss validasi yang mulai bisa menyamai dengan nilai akurasi training dan loss training. Artinya dengan performa model kali ini sudah bisa lebih general dalam melakukan prediksi gambar, tidak hanya menghafal data training.

Dikarenakan performa model yang cukup baik, saya mencoba untuk menambahkan epoch pada model ini untuk melihat kemampuan yang bisa di raih oleh model ini. Saya melakukan training dengan model yang sama ini dengan penambahan 100 epoch (total 200 epoch), dan mendapatkan hasil seperti ini :



Model Mengalami kenaikan validasi akurasi sebesar 1.31% (88.17% menjadi 89.48%) dan penurunan validasi loss sekitar 0.02 (0.350 menjadi 0.332). Dikarenakan CIFAR-10 memiliki dataset yang cukup rumit untuk digeneralisasi maka dari itu model yang saya gunakan juga akan memiliki performa yang cukup lambat.

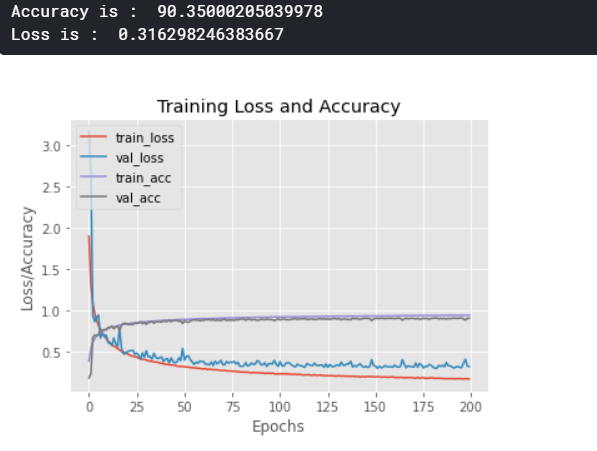
Saya ingin mencoba untuk meningkatkan prefroma daripada model yang saya buat ini, maka dari itu saya mencoba untuk memperkecil persentase dari dropout layer. Saya melakukan hal ini karena, menurut saya persentase dropout di setiap blok terlalu besar untuk model yang memiliki trainable parameter sebanyak 1,343,146 parameter.



Maka dari itu saya merubah layer dropout menjadi 20% di blok pertama, 20% di blok kedua, dan 40% di blok ke tiga. Model akan berubah menjadi seperti berikut :



Kemudian melakukan complie dan training dengan optimizer yang masih sama (Adam) dan 200 epoch. Berikut merupakan hasil dari training :



Model mengalami peningkatan akurasi validasi menjadi 90.35% dan loss validasi menjadi 0.316. Bisa dilihat dengan adanya pengurangan dropout layer akurasi training dapat sedikit mengalami peningkatan beriringan dengan loss training yang mengalami penurunan, hal ini terjadi dikarenakan persentase dropout layer yang berkurang sehingga tidak banyak neuron pada jaringan yang tidak ter“*drop-out*” / terbuang dan training akan berjalan lebih optimal.

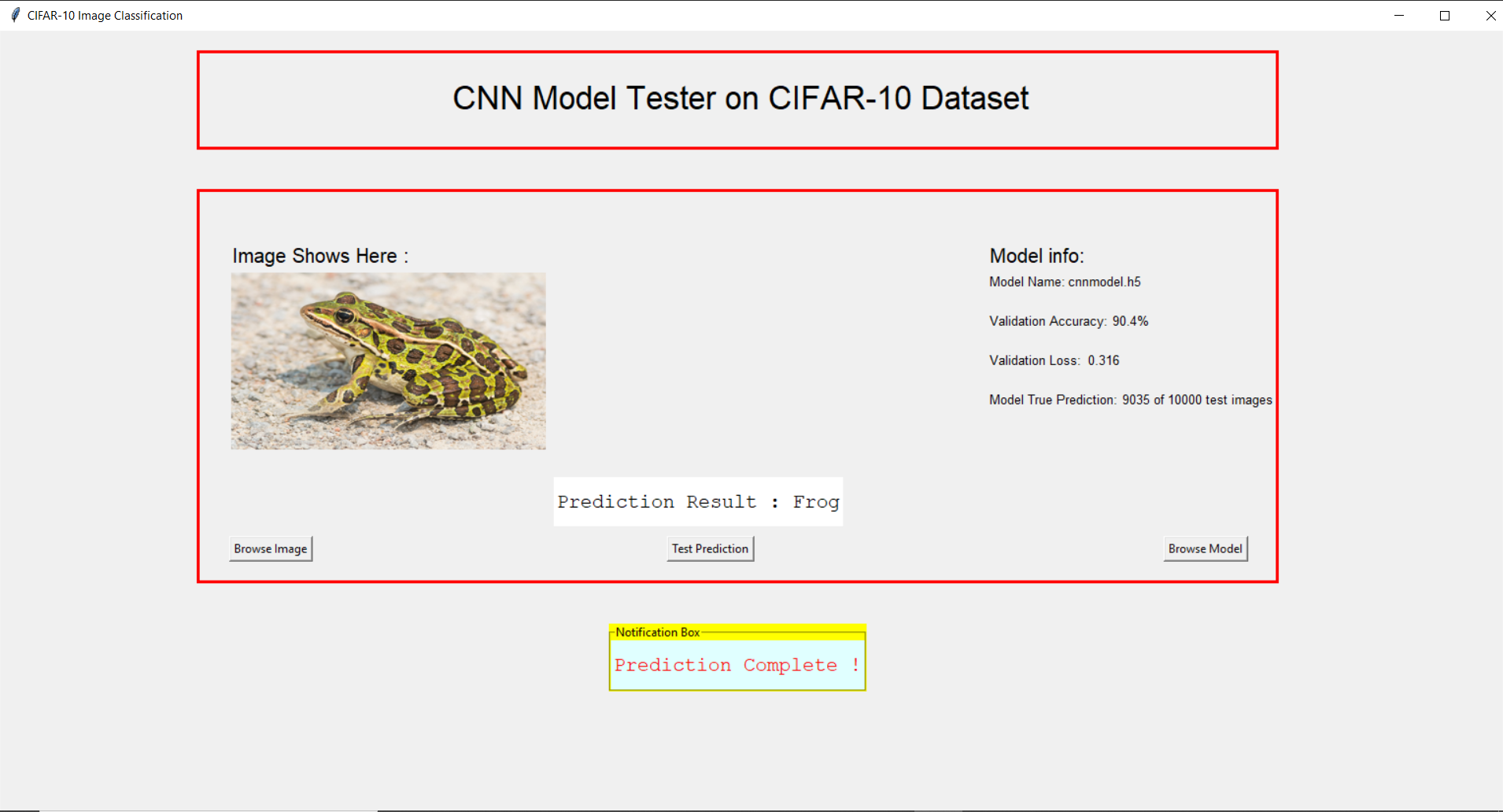
Kemudian saya melakukan saving model menggunakan ***model.save()*** yang sudah disediakn oleh keras.



Dengan melakukan hal ini maka model yang baru saja dijalankan trainingnya akan di simpan dengan format ***.h5***, hal ini di lakukan supaya tidak perlu untuk melakukan training ulang setiap ingin menggunakan model tersebut.

**GUI menggunakan Tkinter**

Saya menggunakan Tkinter sebagai GUI (Graphic User Interface) sederhana untuk mempermudah penggunaan testing untuk model yang dibuat sebelumnya, berikut merupakan tampilan GUI yang dibangun :



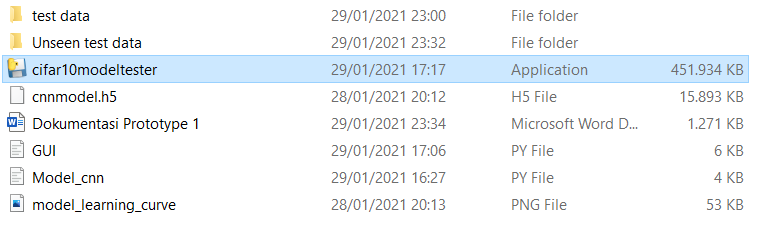
Data untuk prediksi

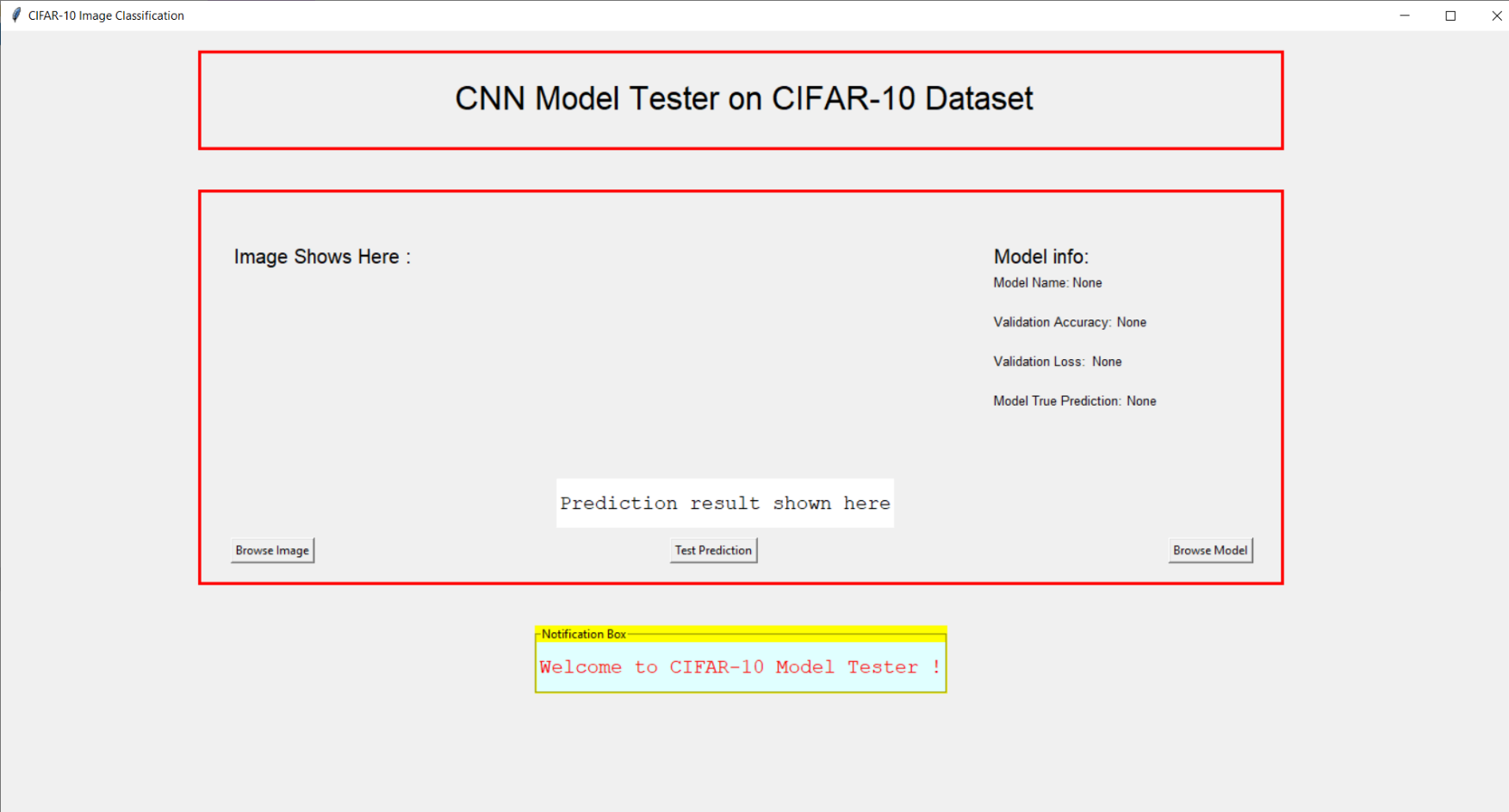


Terdapat 2 folder “test data” dan “unseen test data” yang dimana “test data” merupakan data testing yang terdapat di dalam dataset CIFAR-10, sedangkan “unseen test data” merupakan data yang saya download dari internet dan belum pernah “dilihat” sama sekali oleh model yang akan di tes. Tujuan melakukan testing pada “unseen” data adalah untuk melihat apakah model yang sudah di bangun dapat lebih general dalam memprediksi gambar (model tidak menghafal dalam proses training berlangsung).

Cara penggunaan aplikasi pada pengguna akhir / ***end user***

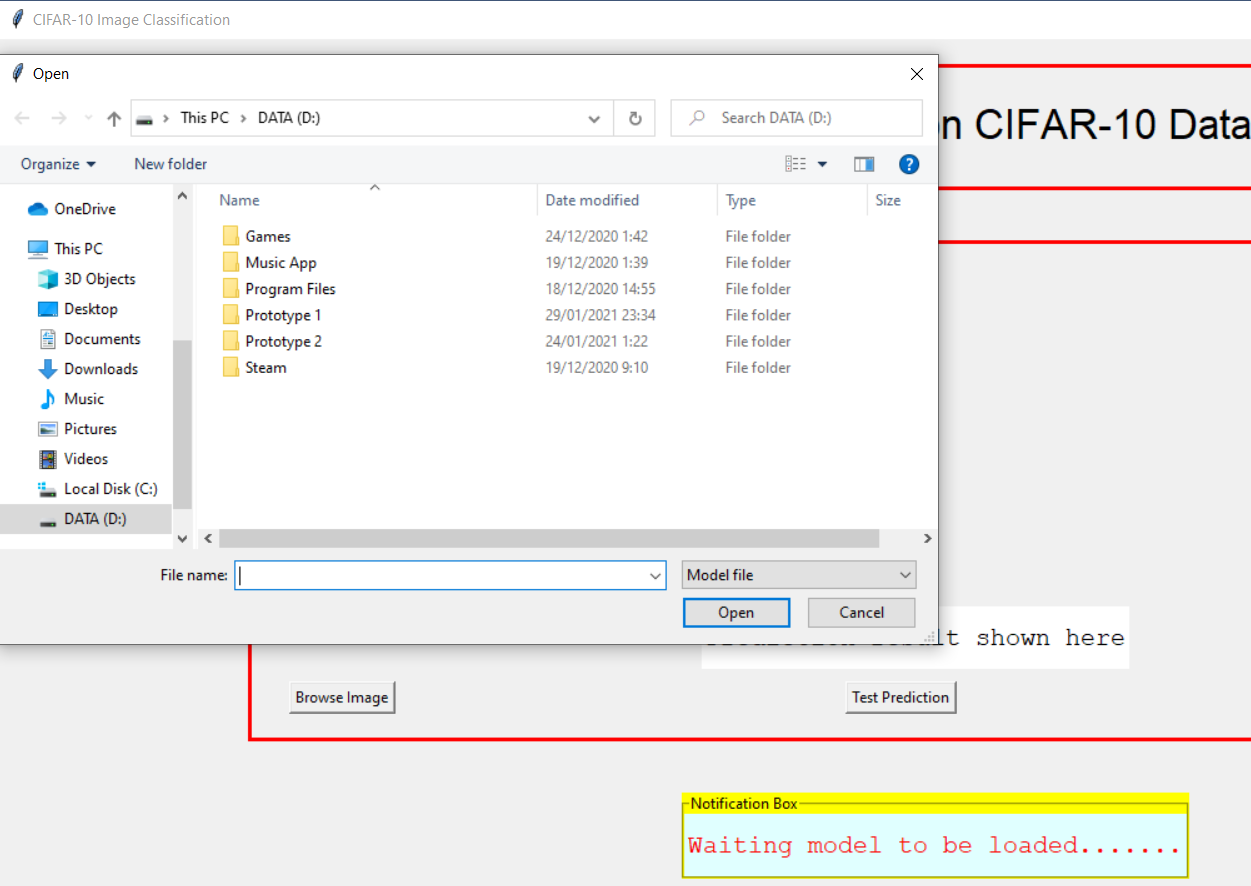
1. Langkah awal, klik dua kali pada icon “***cifar10modeltester.exe***”



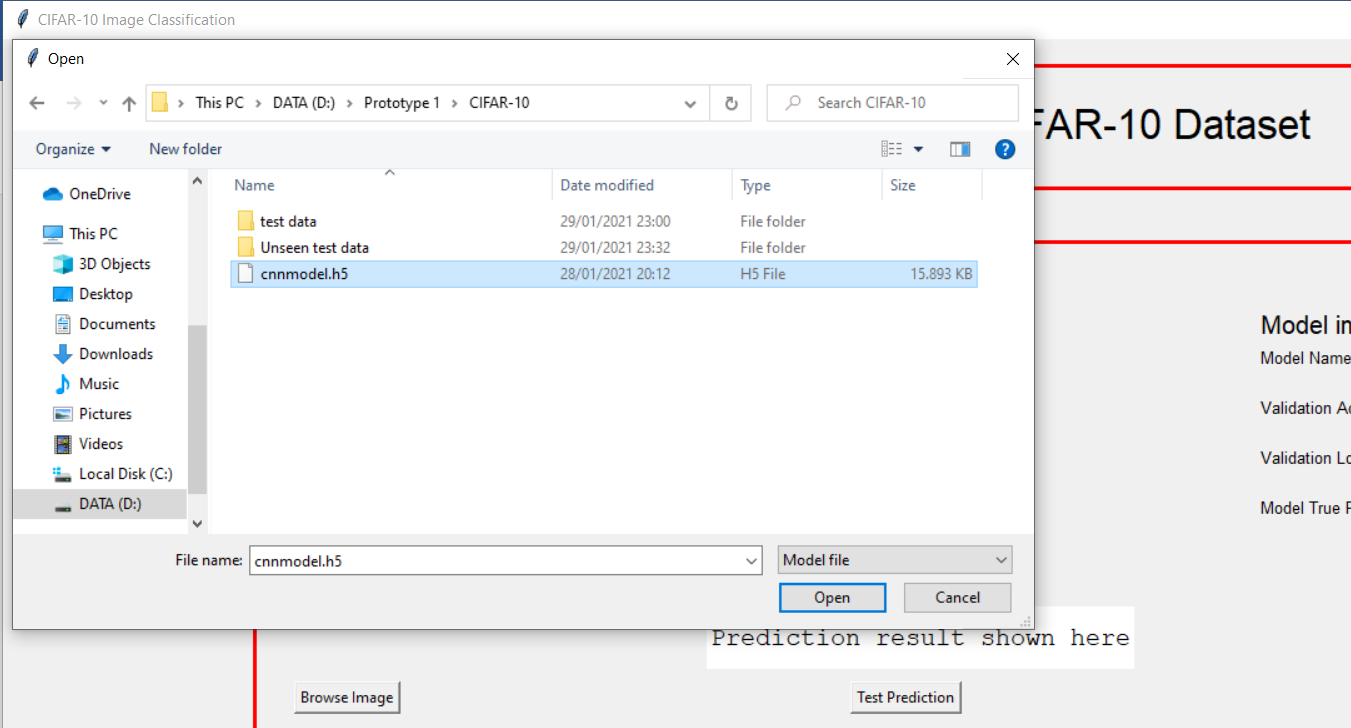
1. Kemudian tampilan awal akan seperti berikut :

Pada halaman awal akan menampilkan judul dari aplikasi ini (Frame yang di atas), informasi dari image yang di upload, model yang digunakan, box yang merupakan hasil dari prediksi (Mid Frame), dan 3 tombol untuk melakukan upload image (Browse Image), upload model (Browse Model), dan melakukan prediksi (Test Prediction). Notification box (Bottom Frame) merupakan kotak notifikasi saat user melakukan interaksi pada button yang tersedia.

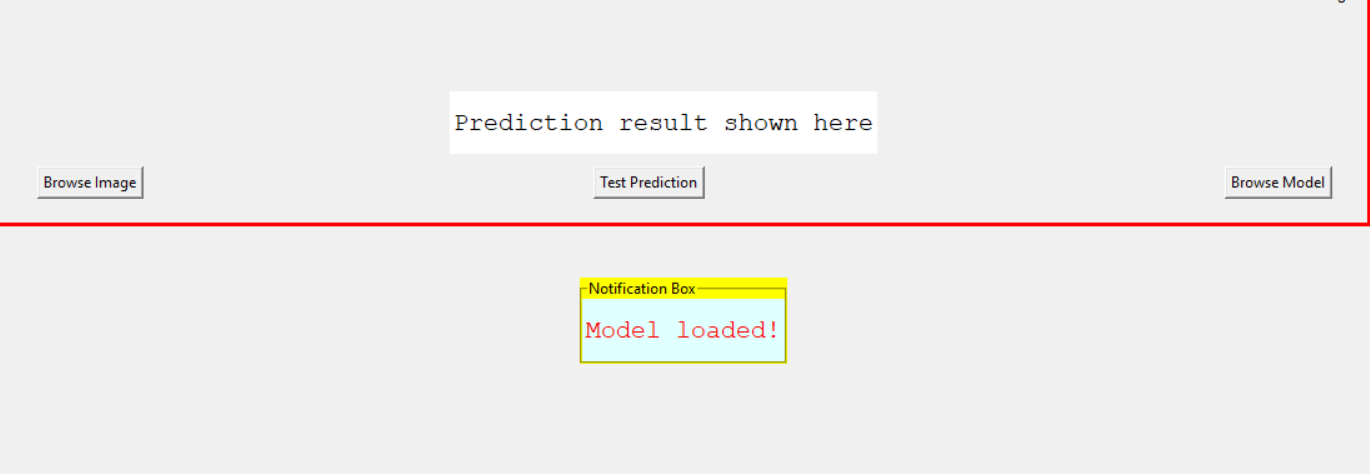
1. Kemudian user bisa memilih untuk mengklik button “browse image” atau “browse Model” untuk mengupload image atau model CNN terlebih dahulu. Pada contoh kali ini saya akan melakukan “Browse Model” terlebih dahulu.



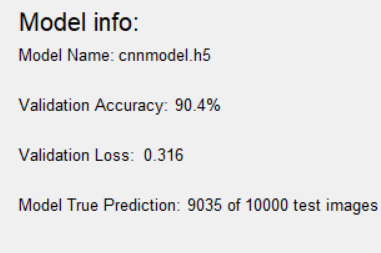
Pada saat user mengklik tombol “Browse Model” maka pop – up seperti gambar di atas akan muncul, dan hal yang harus dilakukan adalah memilih model CNN untuk di upload (ekstensi ***.h5***) dan klik ***open*** / klik dua kali.



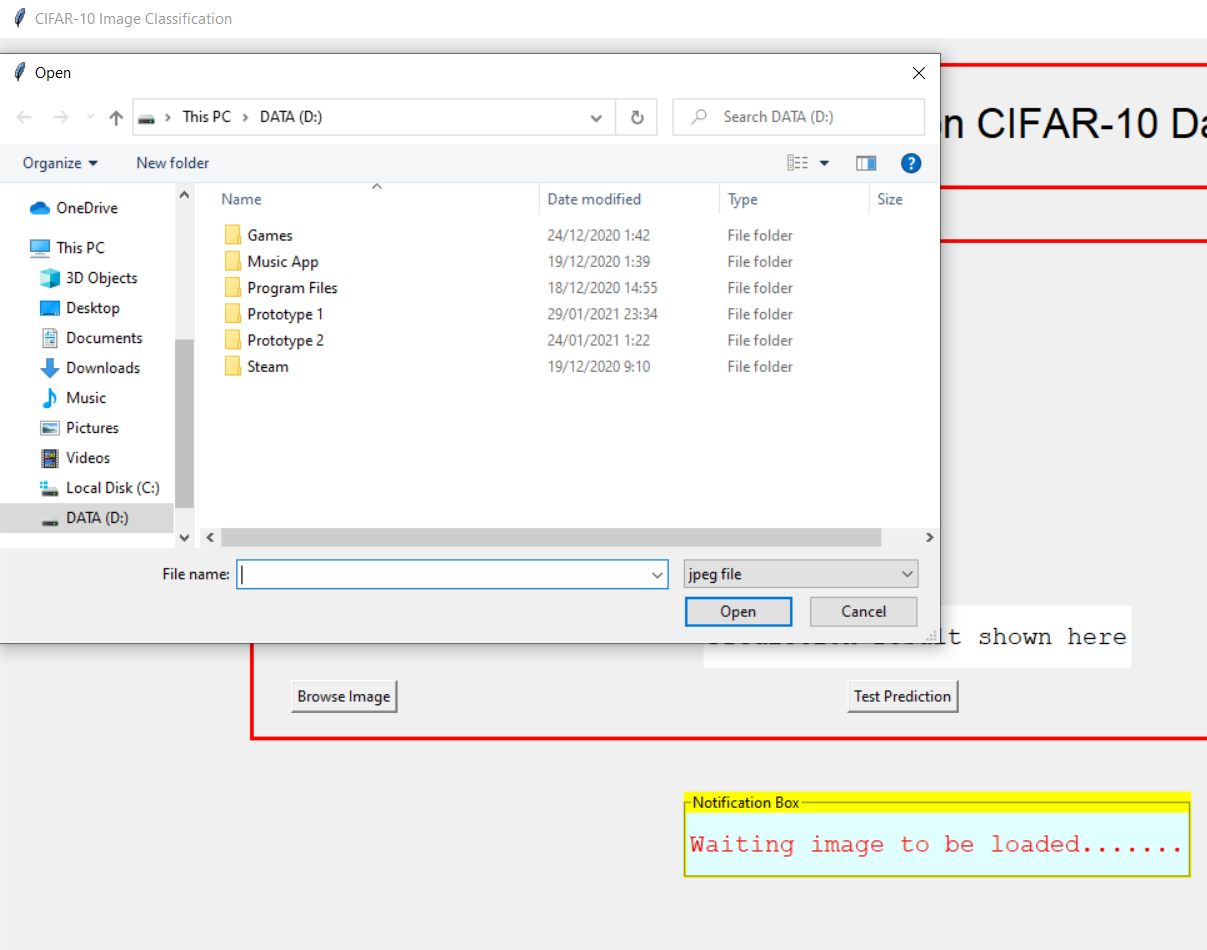
Kemudian kotak notifikasi akan tertulis “Model Loaded !” yang berarti model sudah selesai terload kedalam aplikasi.

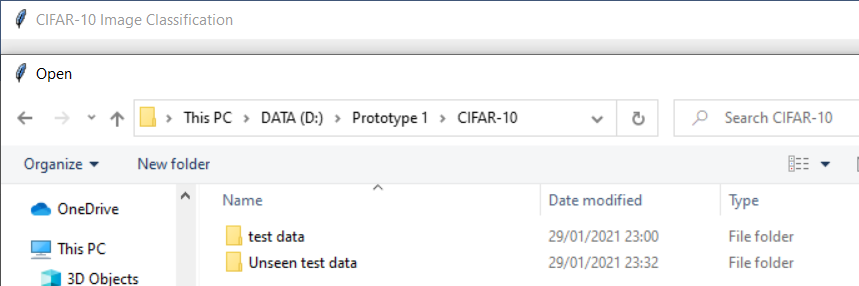


Informasi dari model yang digunakan akan muncul seperti berikut

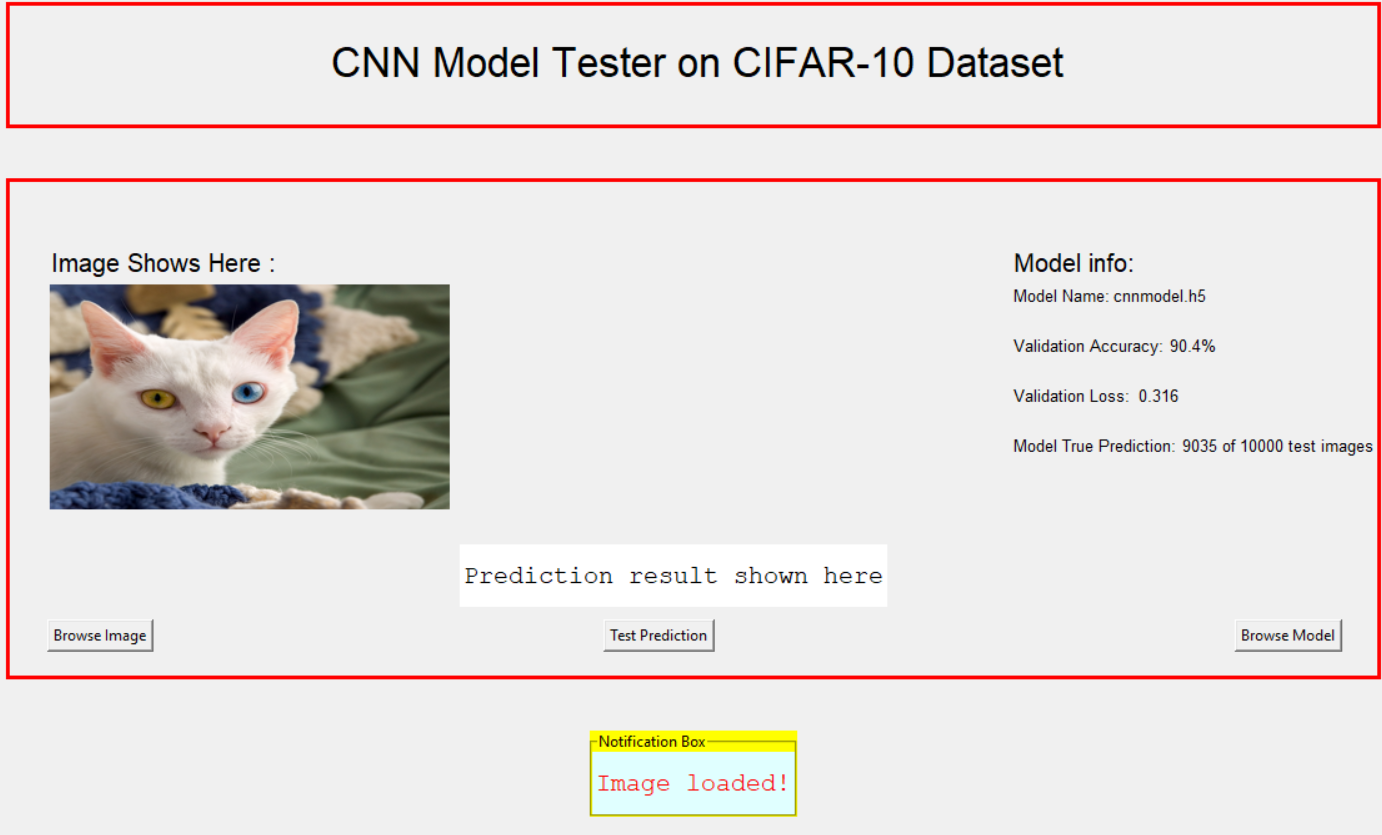


1. Kemudian kita akan mengupload image dengan cara mengklik tombol “Browse Image” dan memilih gambar sama seperti saat kita memilih model CNN (gambar dengan ekstensi ***png*** dan ***jpg*** saja yang dapat di upload).

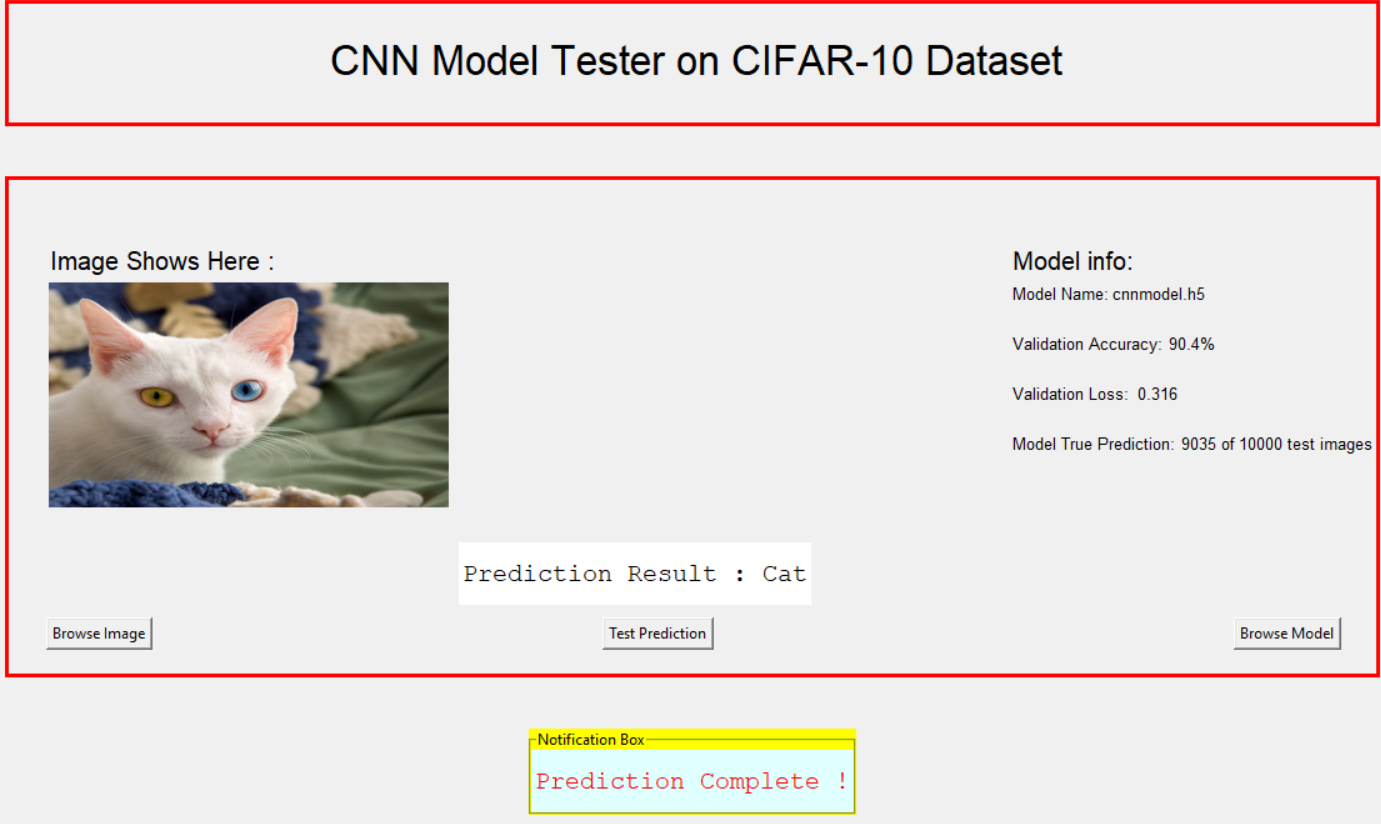


Kemudian pilih salah satu folder berikut dan pilih gambar yang akan di uji / tes.

Setelah memilih gambar kotak notifikasi akan tertulis “Image Loaded !” dan tampilan akan terlihat seperti berikut



1. Langkah terakhir user hanya perlu mengklik tombol “Test Prediction” untuk melakukan prediksi



Kotak prediksi akan berubah dari ***“Prediction result shown here”*** menjadi ***“Prediction Result : + [Predicted image]”***.