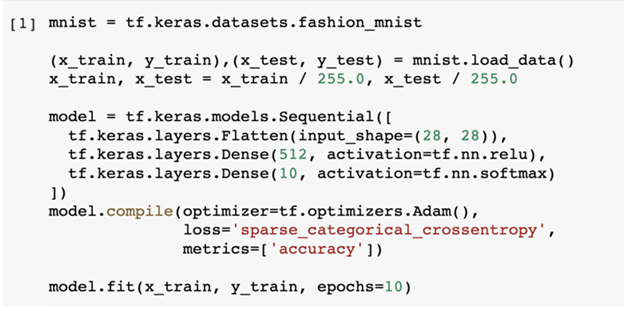
Keras adalah API untuk mengembangkan jaringan saraf tiruan. Dengan Keras kita dapat membuat sebuah *multi layer perceptron* dan*convolutional neural network* dengan sangat mudah. Aplikasi dari Keras sangat luas di mana kita dapat membangun jaringan saraf tiruan untuk klasifikasi gambar, pemrosesan bahasa alami, pengenalan suara, dan prediksi *time series*.

Komponen inti pembangun sebuah jaringan saraf tiruan dalam Keras adalah *layer*. Sebuah layer pada Keras, sama dengan sebuah layer pada MLP yang memiliki beberapa perseptron.

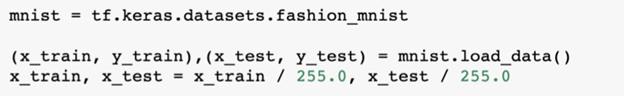
Pada Keras misalnya, kita ingin melakukan klasifikasi pada dataset fashion MNIST seperti contoh di bawah. Dataset Fashion MNIST memiliki label 10 kelas yang terdiri dari baju, sepatu, tas dan sebagainya. Dataset ini berguna untuk mengklasifikasikan sebuah objek fashion.



Dengan Keras, kita dapat membuat sebuah model ML hanya dengan beberapa baris kode seperti di bawah.



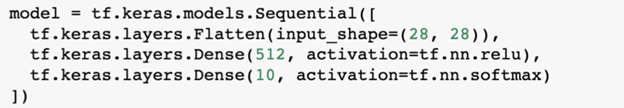
Mari kita bahas kode di atas satu per satu. Hal yang paling pertama adalah kita perlu mempersiapkan data kemudian membaginya menjadi data latih dan data uji. Data fashion MNIST bisa kita dapatkan dengan mudah dari library datasets yang disediakan Keras.



Dalam klasifikasi gambar, setiap piksel pada gambar memiliki nilai dari 0 sampai 255. Kita perlu melakukan normalisasi dengan membagi setiap pixel pada gambar dengan 255. Dengan nilai yang telah dinormalisasi, jaringan saraf dapat belajar dengan lebih baik.



Pada langkah berikutnya kita mendefinisikan arsitektur dari jaringan saraf yang akan kita latih.



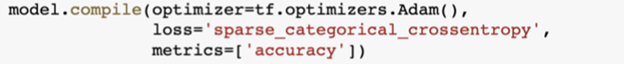
Sama dengan yang kita pelajari pada modul sebelumnya, guna membuat sebuah MLP kita hanya perlu mendefinisikan sebuah input layer, hidden layer dan sebuah output layer. Untuk membuat sebuah model MLP di Keras kita bisa memanggil fungsi tf.keras.models.Sequential([...]) dan menampungnya pada sebuah variabel. Model sequential pada keras adalah tumpukan layer-layer, yang sama seperti pada sebuah MLP.

Berikut kita mendefinisikan 3 layer utama pada model sequential:

* *Input layer* : Adalah layer yang memiliki parameter ‘input\_shape’. Input\_shape sendiri adalah resolusi dari gambar-gambar pada data latih. Dalam hal ini sebuah gambar MNIST memiliki resolusi 28x28 pixel. Sehingga input *shape-*nya adalah (28, 28). Sebuah layer Flatten pada Keras akan berfungsi untuk meratakan input. Meratakan di sini artinya mengubah gambar yang merupakan matriks 2 dimensi menjadi larik 1 dimensi. Pada kasus kita sebuah gambar MNIST yang merupakan matriks 28x 28 elemen akan diubah menjadi larik/array satu dimensi sebesar 784 elemen.
* *Hidden layer* : Dense layer pada Keras merupakan layer yang dapat dipakai sebagai hidden layer dan output layer pada sebuah MLP. Parameter unit merupakan jumlah perseptron pada sebuah layer. Masih ingat bukan, bahwa  activation function adalah fungsi aktivasi yang telah kita pelajari pada modul 5?  Kita dapat menggunakan fungsi aktivasi relu (*rectified linear unit*) atau fungsi aktivasi lain untuk hidden layer kita.
* *Output layer* : Ia didefinisikan dengan membuat sebuah Dense layer. Jumlah unit menyesuaikan dengan jumlah label pada dataset. Untuk fungsi aktivasi pada layer output, gunakan fungsi aktivasi *Sigmoid* ketika hanya terdapat 2 kelas/label pada dataset. Untuk dataset yang memiliki 3 kelas atau lebih, gunakan fungsi aktivasi Softmax. Fungsi aktivasi softmax akan memilih kelas mana yang memiliki probabilitas tertinggi. Untuk data fashion MNIST kita akan menggunakan fungsi aktivasi softmax karena terdapat 10 kelas.

Setelah membuat arsitektur dari MLP, model kita belum bisa melakukan apa-apa. Agar model bisa belajar, kita perlu memanggil fungsi compile pada model kita dan menspesifikasikan optimizer dan loss function. Hal ini sama seperti penjelasan dari propagasi balik pada modul sebelumnya.

Untuk optimizer kita bisa menggunakan Adam. Selanjutnya untuk loss function kita dapat menggunakan sparse categorical entropy pada kasus klasifikasi 3 kelas atau lebih. Untuk masalah 2 kelas, loss function yang lebih tepat adalah binary cross entropy. Parameter metrics berfungsi untuk menampilkan metrik yang dipilih pada proses pelatihan model.



Setelah membuat arsitektur MLP dan menentukan optimizer serta loss functionnya, kita dapat melatih model kita pada data training. Parameter epoch merupakan jumlah berapa kali sebuah model melakukan propagasi balik.



Begitulah bagaimana kita membuat sebuah jaringan saraf dengan Keras. Sangat mudah, bukan?

Pada submodul-submodul selanjutnya kita akan belajar menggunakan data yang sedikit lebih kompleks dari Kaggle.