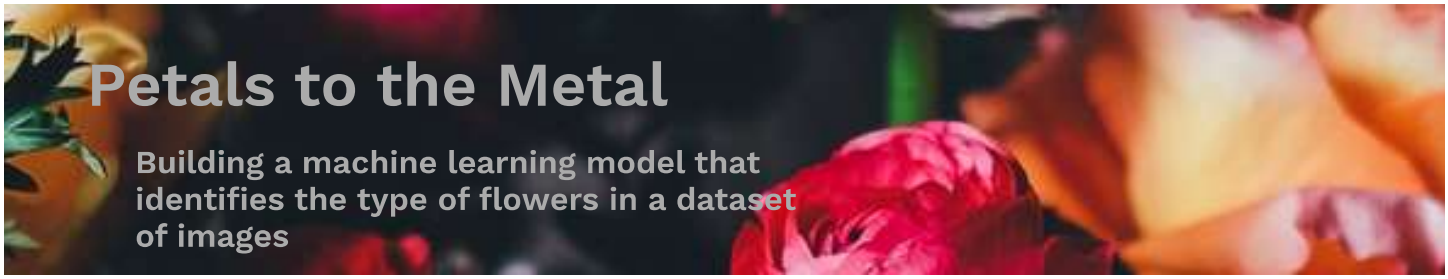
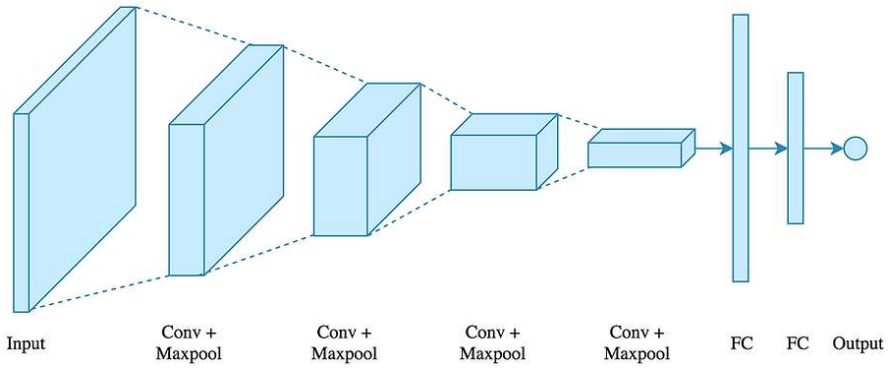


View Notebook

CNN- Convolutional Neural Network

CNN is a subset of neural networks which is mainly used for image classification. CNN has multiple layers, and each layer helps the model understand different features such as shape, vertical height, texture etc. At the end, using all the features extracted the model predicts what class the image could belong to. Since CNN is a supervised learning model, it learns from the error and tries to reduce erroneous classification.

A CNNs architecture consists of multiple types of layers such as Convolution layer, Pooling layer and Fully connected layer. A Convolution layer is the heart of CNN, it takes the input image and applies filters/kernel and produces feature maps. These feature maps are characteristics of the image such as edges, shapes etc. To produce a feature map the filter moves across the image block by block and calculates how good of the fit there is between image and the filter.
Pooling layer is used to downsize the image. The filter for pooling layer moves across the image and assigns a single value to the subregion. This reduces the image size and helps increase computational speed.
The fully connected layer end is a concluding layer for CNN. It aggregates all information and probability of the image belonging to a class.



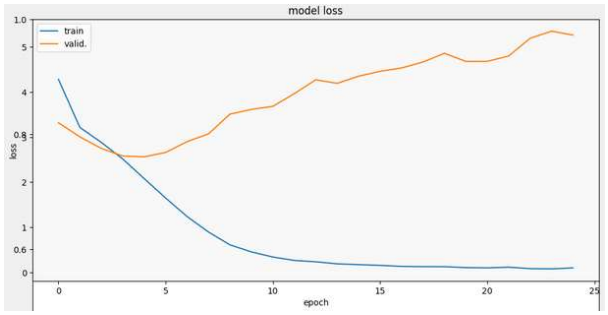
Building CNN model for Image Classification

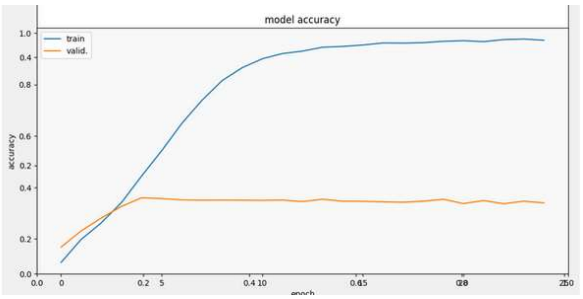
We will be building and comparing CNN models using TPU accelerator for the Petals to the Metal - Flower Classification on TPU | Kaggle competition. I followed A Simple Petals TF 2.2 notebook | Kaggle_ for detecting and loading the data. I will be using adam optimizer to update the weights during training, and sparse_categorical_crossentropy function to calculate loss. I will be running each model for 25 epochs.

Model 1: A simple CNN model

I created a simple CNN model consisting of 3 convolution layer 3 pooling layer and 2 fully connected layer.

```
# Define the model as a sequential sequence of layers
model1 = Sequential()

# Define convolutional layers
model1.add(Conv2D(64, (3, 3), activation='relu', padding='same', input_shape=(*IMAGE_SIZE, 3)))
model1.add(MaxPooling2D((2, 2)))

model1.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
model1.add(MaxPooling2D((2, 2)))

model1.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
model1.add(MaxPooling2D((2, 2)))


# Define classification layers
model1.add(Flatten())
model1.add(Dense(1024, activation='relu'))
model1.add(Dropout(0.5))
model1.add(Dense(len(CLASSES), activation='softmax'))
```



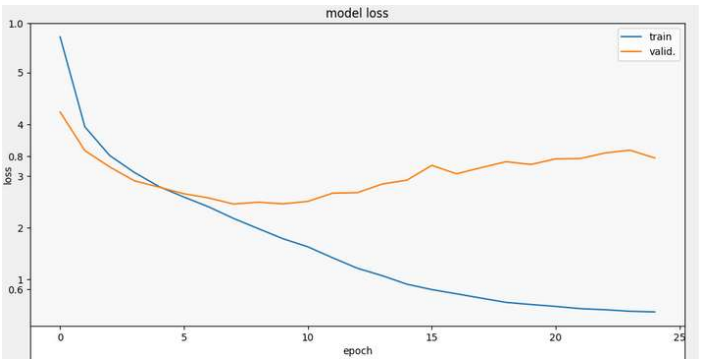


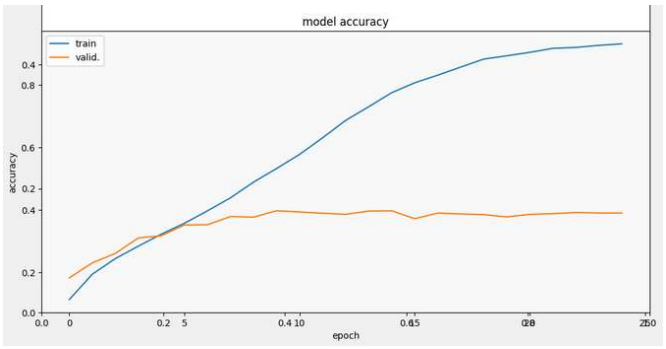
This model after training had training loss of 0.0998 training accuracy: 0.9719 - validation loss: 5.2653 - validation accuracy: 0.3394. The model appears to be overfitted. To resolve the problem of Overfitting we can use regularization which penalizes a complex model being formed during training and ends up creating a more simple model.

Model 1- with Regularization

I added L2 regularization to the hidden layers in model 1.

```
# Define the model as a sequential sequence of layers
model1_regularization = Sequential()

# Define convolutional layers
model1_regularization.add(Conv2D(64, (3, 3), activation='relu', padding='same', input_shape=(*IMAGE_SIZE, 3)))
model1_regularization.add(MaxPooling2D((2, 2)))

model1_regularization.add(Conv2D(256, (3, 3), activation='relu',kernel_regularizer=regularizers.l2(l=0.01), padding='same'))
model1_regularization.add(MaxPooling2D((2, 2)))

model1_regularization.add(Conv2D(512, (3, 3), activation='relu', kernel_regularizer=regularizers.l2(l=0.01),padding='same'))
model1_regularization.add(MaxPooling2D((2, 2)))


# Define classification layers
```



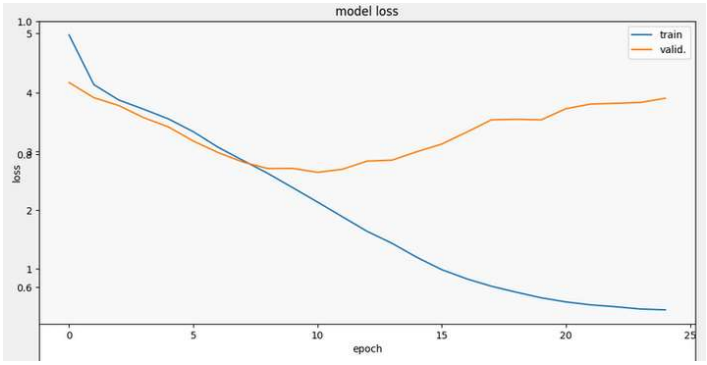


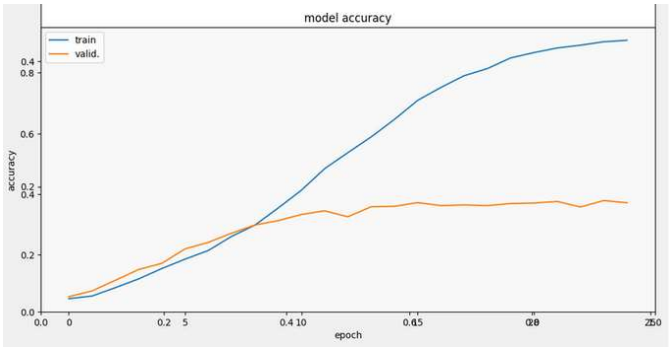
Adding L2 regularization increased the validation accuracy by 4%.

Model 2: more complex model with more convolution layers than model 1

In course to increase the validation accuracy I created a more complex CNN model, which had more convolution layers and pooling layers

```
# Define the model as a sequential sequence of layers
model2 = Sequential()

# Define convolutional layers
model2.add(Conv2D(64, (3, 3), activation='relu', padding='same', input_shape=(*IMAGE_SIZE, 3)))
model2.add(MaxPooling2D((2, 2)))

model2.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
model2.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
model2.add(MaxPooling2D((2, 2)))

model2.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
model2.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
model2.add(MaxPooling2D((2, 2)))

model2.add(Conv2D(1024, (3, 3), activation='relu', padding='same'))
model2.add(Conv2D(1024, (3, 3), activation='relu', padding='same'))
model2.add(MaxPooling2D((2, 2)))

# Define classification layers
model2.add(Flatten())
model2.add(Dense(1024, activation='relu'))
model2.add(Dropout(0.4))
model2.add(Dense(512, activation='relu'))
model2.add(Dropout(0.2))
model2.add(Dense(len(CLASSES), activation='softmax'))
```



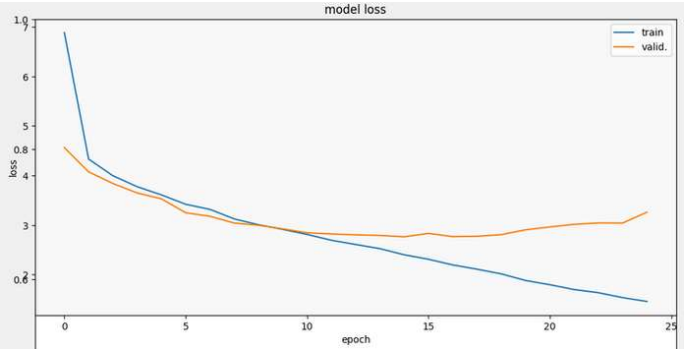


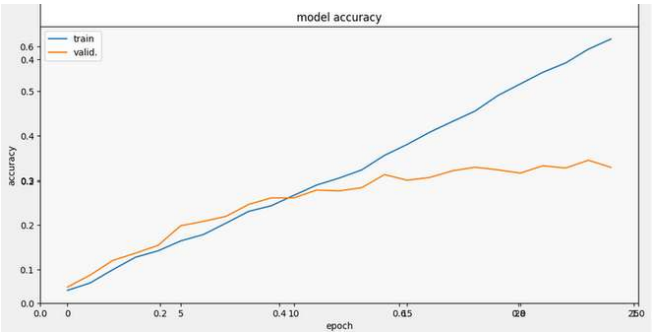
This model had a validation accuracy of 0.3712. This model does slightly better than the model 1.

Model 2- with regularization

Since the model 2 had a very high training accuracy, which could indicate overfitting. I added L2 regularization to the model 2

```
# Define the model as a sequential sequence of layers
model2_regularization = Sequential()

# Define convolutional layers
model2_regularization.add(Conv2D(64, (3, 3), activation='relu', padding='same', input_shape=(*IMAGE_SIZE, 3)))
model2_regularization.add(MaxPooling2D((2, 2)))

model2_regularization.add(Conv2D(256, (3, 3), activation='relu', kernel_regularizer=regularizers.l2(l=0.01), padding='same'))
model2_regularization.add(Conv2D(256, (3, 3), activation='relu', kernel_regularizer=regularizers.l2(l=0.01), padding='same'))
model2_regularization.add(MaxPooling2D((2, 2)))

model2_regularization.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
model2_regularization.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
model2_regularization.add(MaxPooling2D((2, 2)))

model2_regularization.add(Conv2D(1024, (3, 3), activation='relu', kernel_regularizer=regularizers.l2(l=0.01),padding='same'))
model2_regularization.add(Conv2D(1024, (3, 3), activation='relu', kernel_regularizer=regularizers.l2(l=0.01),padding='same'))
model2_regularization.add(MaxPooling2D((2, 2)))


# Define classification layers
model2_regularization.add(Flatten())
model2_regularization.add(Dense(1024, activation='relu'))
model2_regularization.add(Dropout(0.4))
model2_regularization.add(Dense(512, activation='relu'))
model2_regularization.add(Dropout(0.2))
model2_regularization.add(Dense(len(CLASSES), activation='softmax'))
```

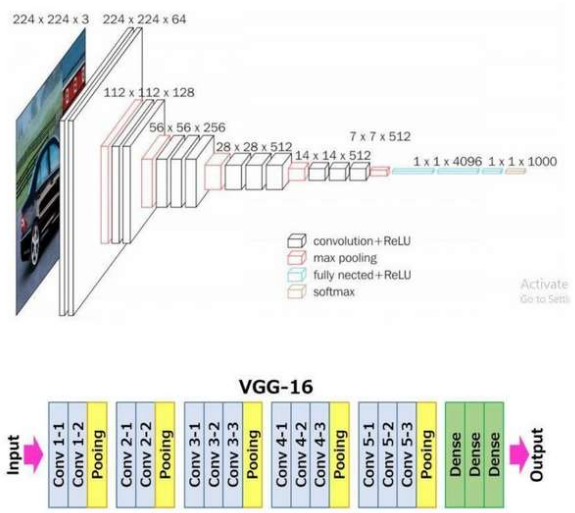




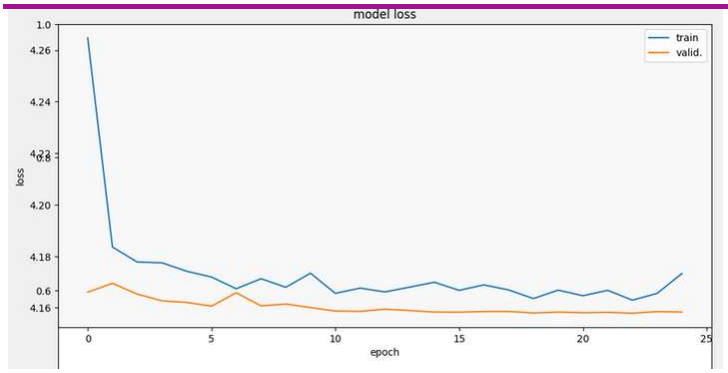
Adding regularization decreased the validation accuracy to 0.3292.

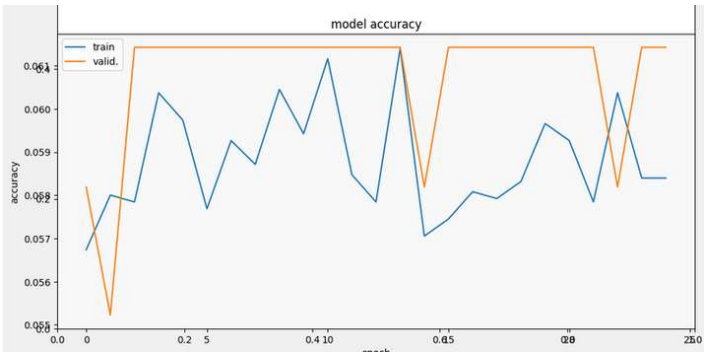
VGG16- without training

VGG16 is considered to be one of the best CNN models for image recognition. I consists of 16 layers.




```
#VGG16
# Define the model as a sequential sequence of layers
vgg16_model = Sequential()

# Define convolutional layers
vgg16_model.add(Conv2D(64, (3, 3), activation='relu', padding='same', input_shape=(*IMAGE_SIZE, 3)))
vgg16_model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
vgg16_model.add(MaxPooling2D((2, 2)))

vgg16_model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
vgg16_model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
vgg16_model.add(MaxPooling2D((2, 2)))

vgg16_model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
vgg16_model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
vgg16_model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
vgg16_model.add(MaxPooling2D((2, 2)))

vgg16_model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
vgg16_model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
vgg16_model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
vgg16_model.add(MaxPooling2D((2, 2)))

vgg16_model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
vgg16_model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
vgg16_model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
vgg16_model.add(MaxPooling2D((2, 2)))

# Define classification layers
vgg16_model.add(Flatten())
vgg16_model.add(Dense(4096, activation='relu'))
vgg16_model.add(Dropout(0.5))
vgg16_model.add(Dense(4096, activation='relu'))
vgg16_model.add(Dropout(0.5))
vgg16_model.add(Dense(len(CLASSES), activation='softmax'))
```

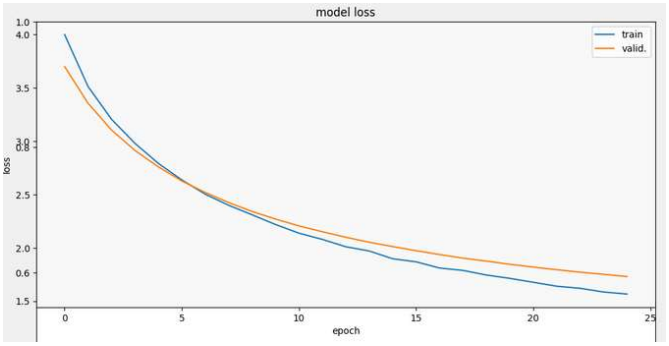


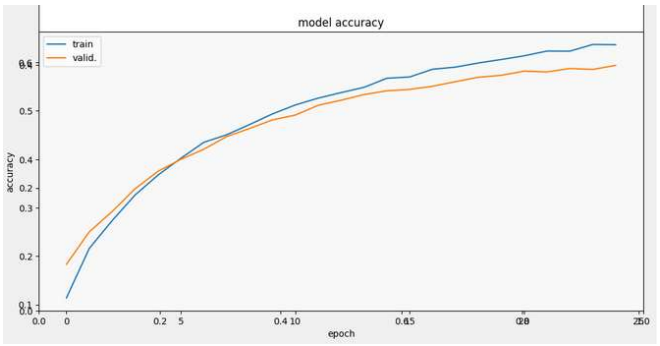


VGG16 model had only 0.0614 validation accuracy. Next up, I used pretrained VGG16 model using ImageNet.

VGG16- pretrained using ImageNet

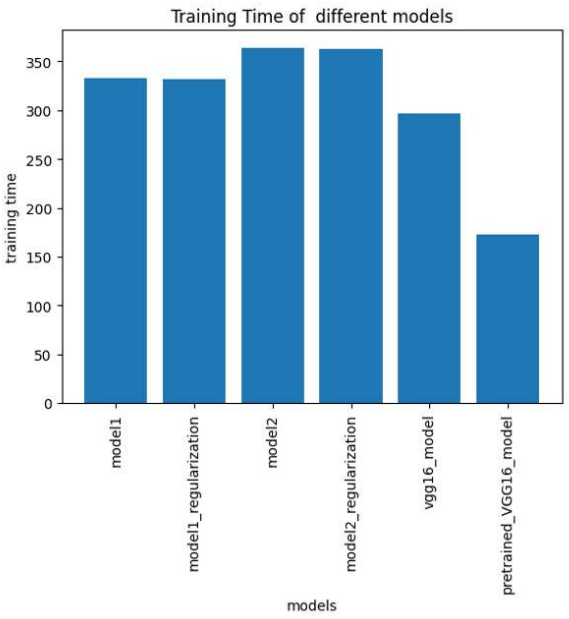
```
pretrained_VGG16 = tf.keras.applications.VGG16(
    weights='imagenet',
    include_top=False ,
    input_shape=[*IMAGE_SIZE, 3]
)
pretrained_VGG16.trainable = False

pretrained_VGG16_model = tf.keras.Sequential([
    # To a base pretrained on ImageNet to extract features from images...
    pretrained_VGG16,
    # ... attach a new head to act as a classifier.
    tf.keras.layers.GlobalAveragePooling2D(),
    tf.keras.layers.Dense(len(CLASSES), activation='softmax')
])
```



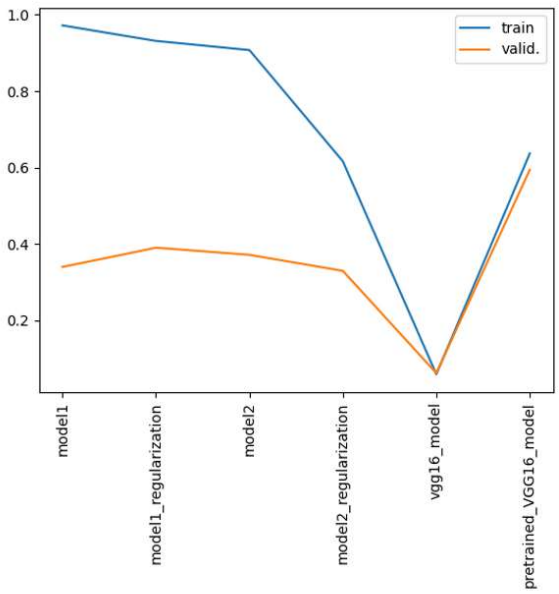


Pretrained VGG16 model had the best validation accuracy amongst all the model above.

Model Comparison



Training Time vs Model plot



Validation Accuracy vs Model plot

The training time for model 2 is the highest as it has more layers. Pretrained VGG16 Model has highest validation accuracy. The next best model is the model1 with regularization. Submitting the predictions from model 1 with regularization gave accuracy of 0.27318 on the test data.

My contributions

I started by creating a simple CNN model and found that it was overfitting. To overcome this I used L2 regularization. I also created a more complex model with more layers and found that it did in fact increase the accuracy, but on applying regularization the accuracy decreased. I also implemented the VGG16 model with and without pretraining. I also compared the training time and accuracy of the model created. Pretrained VGG16 model had the highest accuracy while having a low training time.

Email
axc5981@mavs.uta.edu

Follow Me

References

1. Everything you need to know about VGG16 | by Great Learning | Medium
2. Binary Classification Using Convolution Neural Network (CNN) Model | by Mayank Verma | Medium
3. A Simple Petals TF 2.2 notebook | Kaggle
4. https://www.analyticsvidhya.com/blog/2020/09/overfitting-in-cnn-show-to-treat-overfitting-in-convolutional-neural-networks/