Machine Learning Engineer Nanodegree
Capstone Project
Pragyan Prakash
May 3rd, 2019

I. Definition

1.1 Project Overview

Domain

Botany, also called plant science(s), plant biology or phytology, is the science of plant life and a branch of biology. A **botanist**, plant scientist or phytologist is a scientist who specializes in this field.

Relevant Publications

Nilsback, M-E. and Zisserman, A.
A Visual Vocabulary for Flower Classification
Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (2006)

Nilsback, M-E. and Zisserman, A.
Delving into the whorl of flower segmentation
Proceedings of the British Machine Vision Conference (2007)

Nilsback, M-E. and Zisserman, A.
Automated flower classification over a large number of classes
Proceedings of the Indian Conference on Computer Vision, Graphics and Image Processing (2008)

Dataset

The data that I am using is flower 17 data set.

Download link: wget http://www.robots.ox.ac.uk/~vgg/data/flowers/17/17flowers.tgz

Description

- The dataset consists of 17 category of flower with 80 images for each class.
- The flowers chosen are some common flowers in the UK.
- The images have large scale, pose and light variations and there are also classes with large variations of images within the class and close similarity to other classes.

1.2 Problem Statement

Accurate classification of plant species with a small number of images. This model can be used by botanist to classify flowers.

1.2.1 Strategy

A breakthrough in building models for image classification came with the discovery that a convolutional neural network (CNN) could be used to progressively extract higher- and higher-level representations of the image content. Instead of preprocessing the data to derive features like textures and shapes, a CNN takes just the image's raw pixel data as input and "learns" how to extract these features, and ultimately infer what object they constitute.

To start, the CNN receives an input feature map: a three-dimensional matrix where the size of the first two dimensions corresponds to the length and width of the images in pixels. The size of the third dimension is 3 (corresponding to the 3 channels of a color image: red, green, and blue). The CNN comprises a stack of modules, each of which performs three operations.

1.Convolution

A *convolution* extracts tiles of the input feature map, and applies filters to them to compute new features, producing an output feature map, or *convolved feature* (which may have a different size and depth than the input feature map). Convolutions are defined by two parameters:

- **Size of the tiles that are extracted** (typically 3x3 or 5x5 pixels).
- **The depth of the output feature map**, which corresponds to the number of filters that are applied.

During a convolution, the filters (matrices the same size as the tile size) effectively slide over the input feature map's grid horizontally and vertically, one pixel at a time, extracting each corresponding tile.

For each filter-tile pair, the CNN performs element-wise multiplication of the filter matrix and the tile matrix, and then sums all the elements of the resulting matrix to get a single value. Each of these resulting values for every filter-tile pair is then output in the *convolved feature* matrix

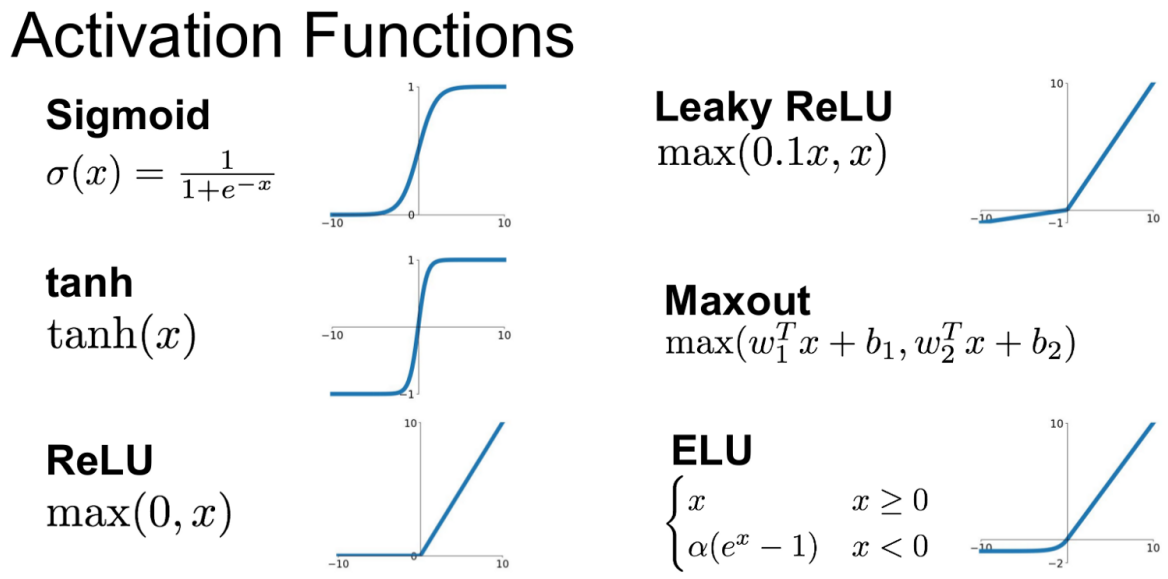
During training, the CNN "learns" the optimal values for the filter matrices that enable it to extract meaningful features (textures, edges, shapes) from the input feature map. As the number of filters (output feature map depth) applied to the input increases, so does the number of features the CNN can extract. However, the tradeoff is that filters compose the majority of resources expended by the CNN, so training time also increases as more filters are added. Additionally, each filter added to the network provides less incremental value than the previous one, so engineers aim to construct networks that use the minimum number of filters needed to extract the features necessary for accurate image classification.

2.Activation

In artificial neural networks, the **activation function** of a node defines the output of that node, or "neuron," given an input or set of inputs. This output is then used as input for the next node and so on until a desired solution to the original problem is found.

It maps the resulting values into the desired range such as between 0 to 1 or -1 to 1 etc. (depending upon the choice of activation function).

Common Activation Functions



ReLU

Following each convolution operation, the CNN applies a Rectified Linear Unit (ReLU) transformation to the convolved feature, in order to introduce nonlinearity into the model. The ReLU function, F(x)=max(0,x), returns *x* for all values of $x > 0$, and returns 0 for all values of $x \leq 0$.

3.Pooling

After ReLU comes a pooling step, in which the CNN downsamples the convolved feature (to save on processing time), reducing the number of dimensions of the feature map, while still preserving the most critical feature information. A common algorithm used for this process is called max pooling.

Max pooling operates in a similar fashion to convolution. We slide over the feature map and extract tiles of a specified size. For each tile, the maximum value is output to a new feature map, and all other values are discarded. Max pooling operations take two parameters:

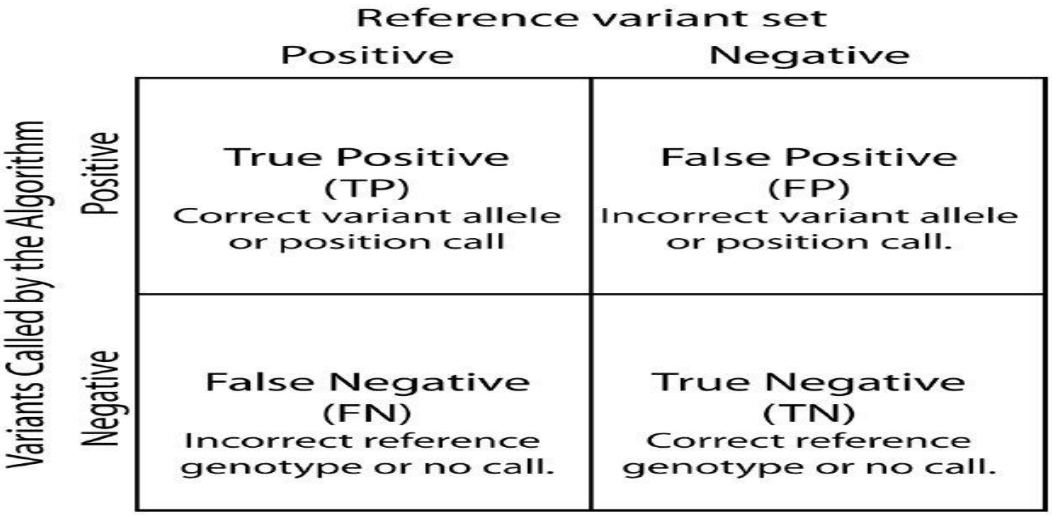
- **Size** of the max-pooling filter (typically 2x2 pixels)
- **Stride**: the distance, in pixels, separating each extracted tile. Unlike with convolution, where filters slide over the feature map pixel by pixel, in max pooling, the stride determines the locations where each tile is extracted. For a 2x2 filter, a stride of 2 specifies that the max pooling operation will extract all nonoverlapping 2x2 tiles from the feature map

Fully Connected Layers

- At the end of a convolutional neural network are one or more fully connected layers (when two layers are "fully connected," every node in the first layer is connected to every node in the second layer).
- Their job is to perform classification based on the features extracted by the convolutions.
- Typically, the final fully connected layer contains a **softmax** activation function, which outputs a probability value from 0 to 1 for each of the classification labels the model is trying to predict.

1.3 Metrics

Accuracy is one metric for evaluating classification models.



$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

Also,

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Where,

- *TP* = True Positives,
- *TN* = True Negatives,
- *FP* = False Positives, and
- *FN* = False Negatives.

2. Analysis

2.1 Data Exploration

The dataset consists of 17 category of flower with 80 images for each class (total 1360 images)

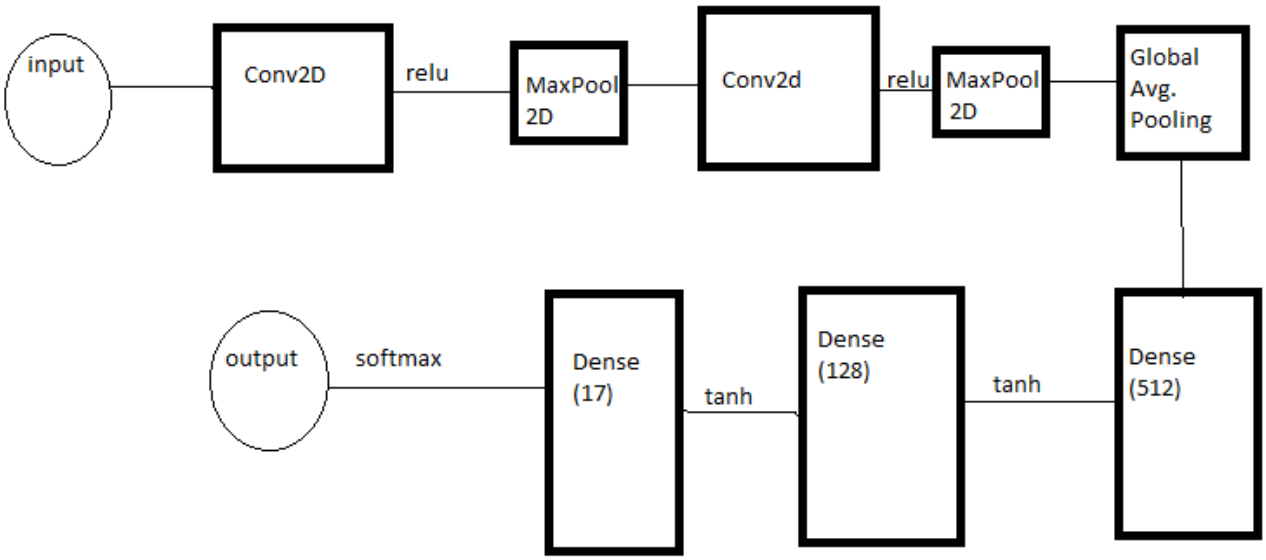
```
plt.subplot(1,2,1)
img = load_img("image/Daffodil/image_0020.jpg",target_size=(64,64))
x = img_to_array(img)/255.
plt.imshow(x)

plt.subplot(1,2,2)
img = load_img("image/Snowdrop/image_0082.jpg",target_size=(64,64))
x = img_to_array(img)/255.
plt.imshow(x)
plt.show()
```

2.2 Algorithms and Techniques

Generally ,CNN architecture for image classification consists of two main parts, “feature extractor” that based on conv-layers, and “classifier” which usually based on fully connected layers.

A CNN works by extracting features from images. This eliminates the need for manual feature extraction. The features are not trained! They’re learned while the network trains on a set of images. This makes deep learning models extremely accurate for computer vision tasks. CNNs learn feature detection through tens or hundreds of hidden layers. Each layer increases the complexity of the learned features.

I have used a CNN network for this task. The network architecture is as follows:



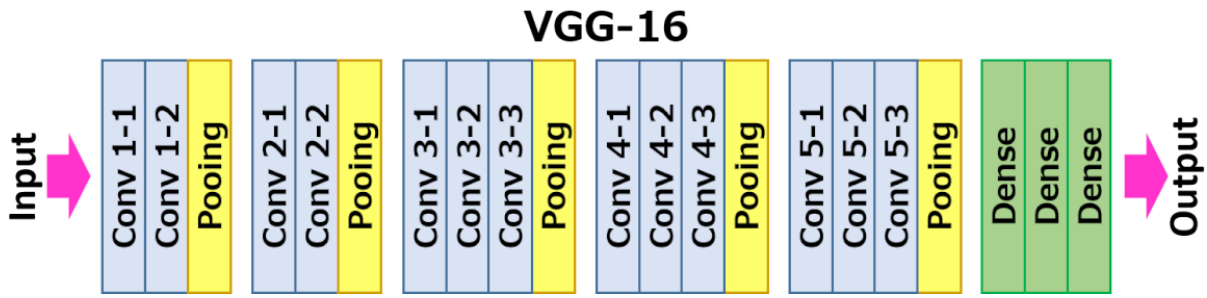
The solution model summary is as follows:-

```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 64, 64, 32)        2432
_________________________________________________________________
activation_1 (Activation)    (None, 64, 64, 32)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 32, 32, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 28, 28, 96)        76896
_________________________________________________________________
activation_2 (Activation)    (None, 28, 28, 96)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 14, 14, 96)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 14, 14, 96)        0
_________________________________________________________________
global_average_pooling2d_1 ( (None, 96)                0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               49664
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               65664
_________________________________________________________________
dropout_4 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 17)                2193
_________________________________________________________________
activation_3 (Activation)    (None, 17)                0
=================================================================
Total params: 196,849
Trainable params: 196,849
Non-trainable params: 0
_________________________________________________________________
```

2.3 Benchmark

I am using pre-trained VGG16 Keras implementation as my benchmark model**. (Training only the prediction head)**



2.4 Benchmark Threshold

- The pre-trained VGG16 model can definitely outperform our solution model if fine-tuned.
- So, only prediction head is trained and it's accuracy i.e 76% is set as the standard threshold.

```
model_benchmark.evaluate(valid_x,valid_y)
```

```
272/272 [==============================] -
```

```
[0.9797674137003281 6, 0.7610294117647058 4]
```

3. Methodology

3.1 Data Preprocessing

- RGB (Red, Green, Blue) are 8 bit each.
  The range for each individual color is 0-255 (as 2^8 = 256 possibilities).
  The combination range is 256*256*256.
- By dividing by 255, the 0-255 range can be described with a 0.0-1.0 range where 0.0 means 0 (0x00) and 1.0 means 255 (0xFF).
- Normalization will help you to remove distortions caused by lights and shadows in an image.

- Using Data argumentation to generalize the model.
- Keras default method is used for argumentation

```
data_generator = ImageDataGenerator(rotation_range=30, width_shift_range=0.1,
                                    height_shift_range=0.1, shear_range=0.2,
                                    zoom_range=0.2,horizontal_flip=True, fill_mode="nearest")
```

3.2 Metrics, Algorithms, and Techniques

I am using keras implementation of Metrics and Algorithms.
The packages are imported as follows:-

```
from keras import layers
from keras.models import Sequential, Model
from keras.preprocessing.image import ImageDataGenerator
from keras.applications.vgg16 import VGG16
from keras.preprocessing.image import load_img, img_to_array
from sklearn.model_selection import train_test_split
from keras.layers import Conv2D, MaxPooling2D, GlobalAveragePooling2D,Dense
from keras.layers import Activation, Flatten, Dropout
from keras.callbacks import ModelCheckpoint
from keras import optimizers
```

The model implementation is as follows :

```
model = Sequential()

model.add(Conv2D(32, (5, 5), padding='same', input_shape=train_x.shape[1:]))
model.add(Activation('relu'))

model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(96, (5, 5)))
model.add(Activation('relu'))

model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

#     model.add(Flatten())
model.add(GlobalAveragePooling2D())

model.add(Dense(512, activation='tanh'))
model.add(Dropout(0.25))

#    model.add(Dense(256, activation='tanh'))
#    model.add(Dropout(0.25))

model.add(Dense(128, activation='tanh'))
model.add(Dropout(0.25))

model.add(Dense(17))
model.add(Activation('softmax'))
```

Afterwards, the model is compiled, the “opt” variable is used to specify the optimizer.

```
opt=optimizers.Adam(lr=0.001)
```

```
model.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accuracy'])
```

Finally, the model is fitted and result is evaluated. The log of loss and accuracy is stored in “hist” variable which is used for plotting the loss vs epochs and accuracy vs epochs graphs.

```
hist=model.fit_generator(data_generator.flow(train_x, train_y, batch_size=150),
                                             steps_per_epoch=189, epochs=60, verbose=2,
                                             validation_data=(valid_x, valid_y))
```

```
model.evaluate(valid_x,valid_y)
```

```
272/272 [==============================]
```

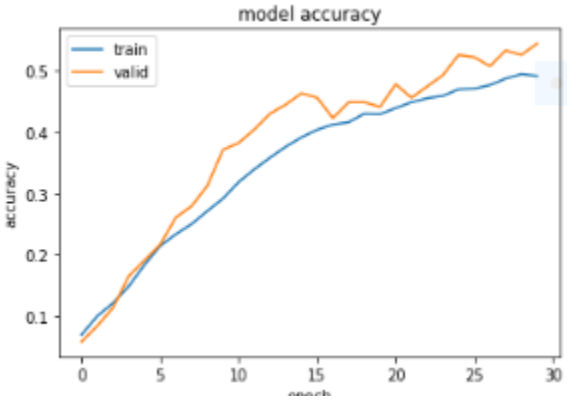
```
[1.0478036482544506, 0.81985294117647056]
```

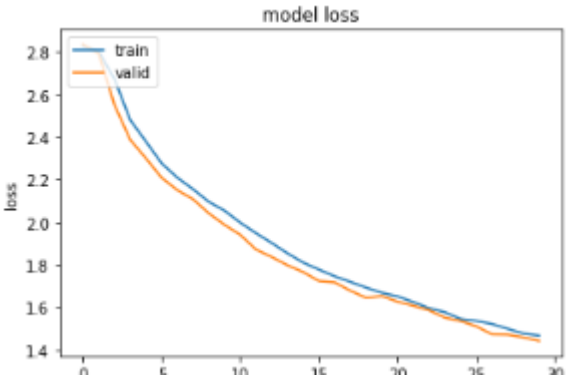
4. Results

4.1 Refinement

EXPERIMENT 1

Overview: • Defaults settings • Learning rate = 0.01 • Batch –size = 100 • epochs=30 • optimizer = SGD	**Observation**: • Training loss seems to be decreasing • Validation loss more or less following training loss. • Test accuracy is around 54% **Analysis**: Maybe learning rate is too high or changing optimizer might help





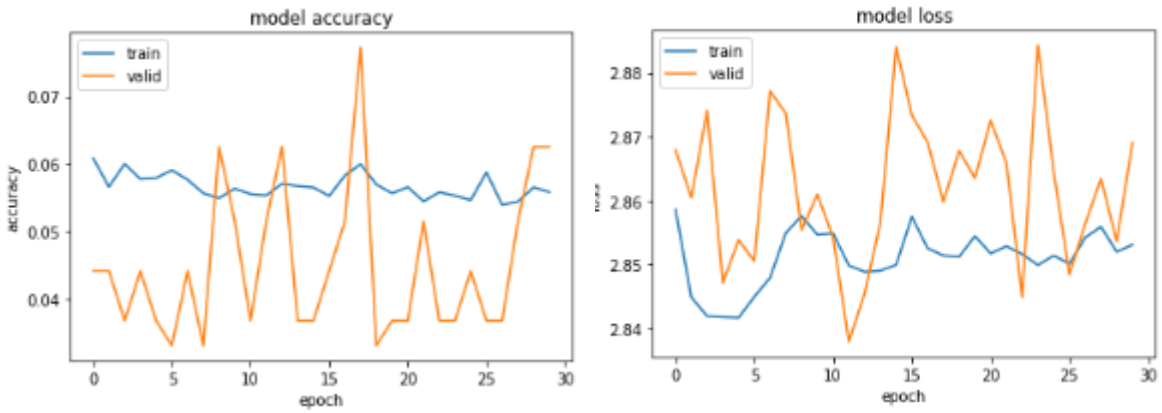

```
model.evaluate(valid_x,valid_y)
```

```
272/272 [==============================] -
[1.4407783957088696, 0.5441176470588348]
```

EXPERIMENT 2

Using **ADAM** as optimizer

Overview:	**Observation**:
• Defaults settings • Learning rate = 0.01 • Batch –size = 100 • epochs=30 • optimizer = ADAM	• highly overfitting • Test accuracy is around 6% **Analysis**: Maybe learning rate is high

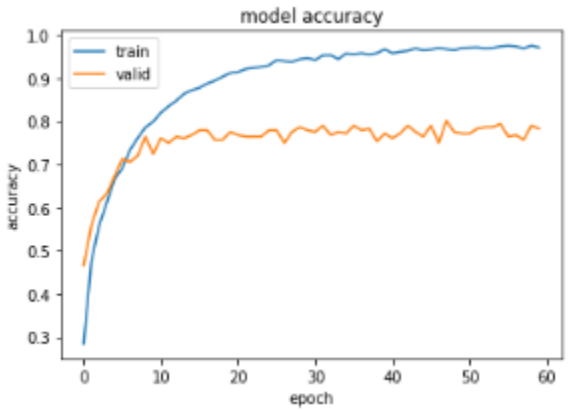


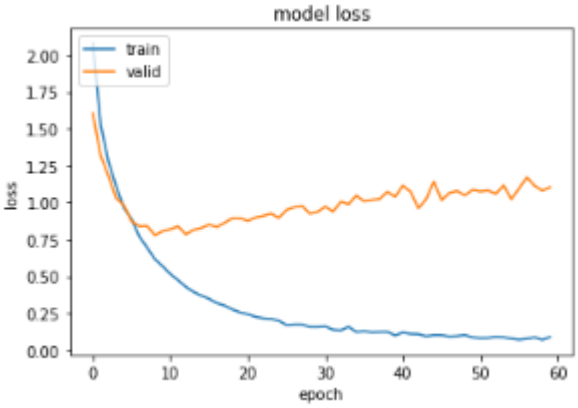

```
model.evaluate(valid_x,valid_y)
```

```
272/272 [==============================]
[2.8690471649169922, 0.0625]
```

EXPERIMENT 3

Overview:	**Observation**:
• Defaults settings • Learning rate = 0.001 • Batch –size = 100 • epochs=60 • optimizer = ADAM	• overfitting is reduced to some extent • Test accuracy is around 78% **Analysis**: Maybe lowering learning rate might help



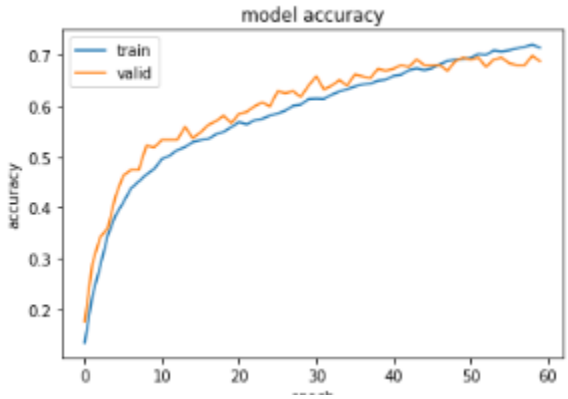


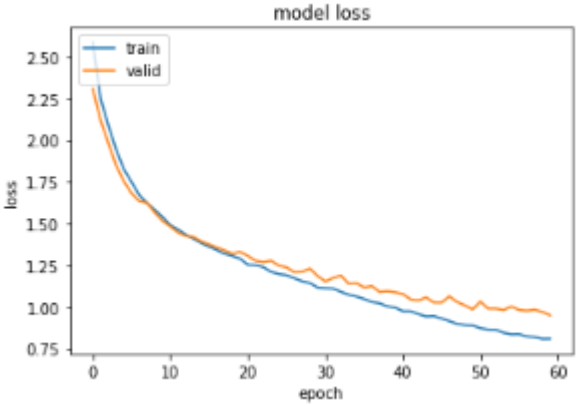

```
model.evaluate(valid_x,valid_y)
```

```
272/272 [==============================]
[1.105215955306502, 0.7830882352941764]
```

EXPERIMENT 4

Overview:	**Observation**:
• Defaults settings • **Learning rate = 0.0001** • Batch –size = 100 • epochs=60 • optimizer = ADAM	• model performance deceased • Test accuracy is around 69% **Analysis**: previous lr=0.001 seems better



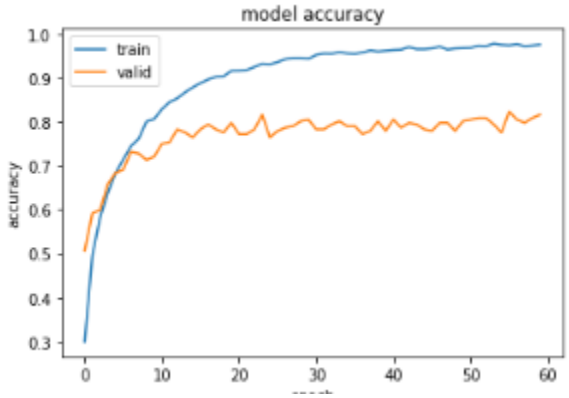


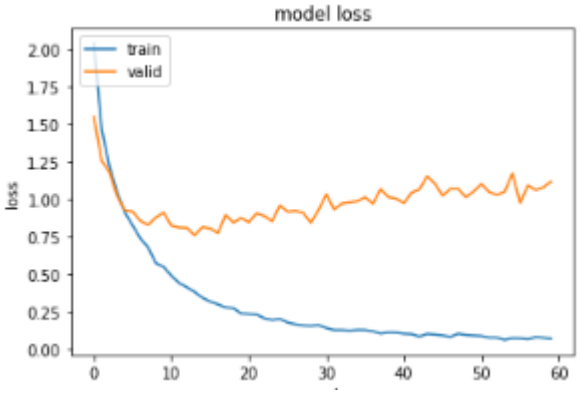

```
model.evaluate(valid_x,valid_y)
```

```
272/272 [==============================] -
[0.91254411024205828, 0.69117647058823528]
```

EXPERIMENT 5

Overview: • Defaults settings • Learning rate = 0.001 • **Batch –size = 150** • epochs=60 • optimizer = ADAM	**Observation**: • overfitting is reduced to some extent • Test accuracy is around 81% **Analysis**: Model still overfits but accuracy is improved






```
model.evaluate(valid_x,valid_y)
```

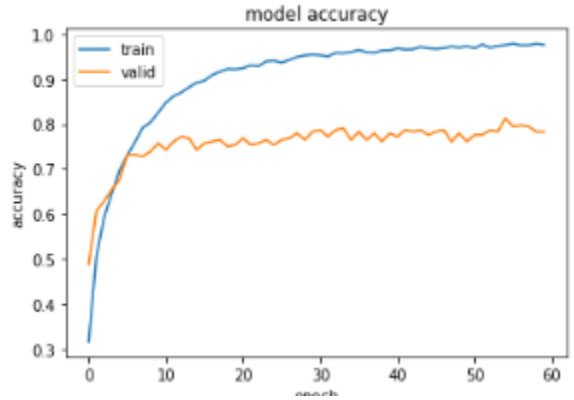
```
272/272 [==============================] -
```

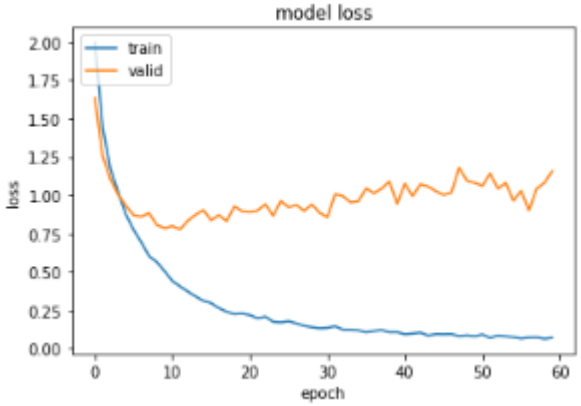
```
[1.0478036482544506, 0.81985294117647056]
```

Clearly, changing the batch size had a significant impact on the model performance. So we'll do one final experiment with batch size as 200

EXPERIMENT 6

Overview:	**Observation**:
• Defaults settings • Learning rate = 0.001 • **Batch –size = 200** • epochs=60 • optimizer = ADAM	• overfitting is observed • Test accuracy is approx. 78 % **Analysis**: No significant change is observed



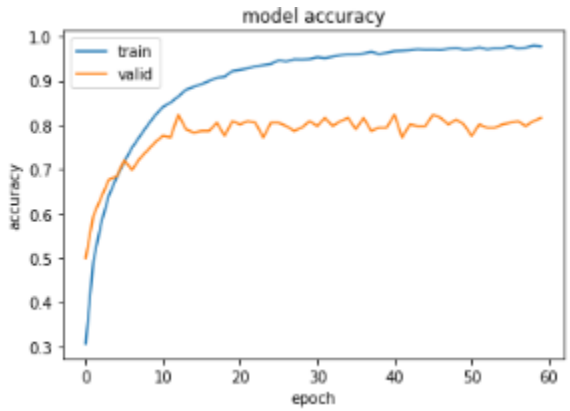


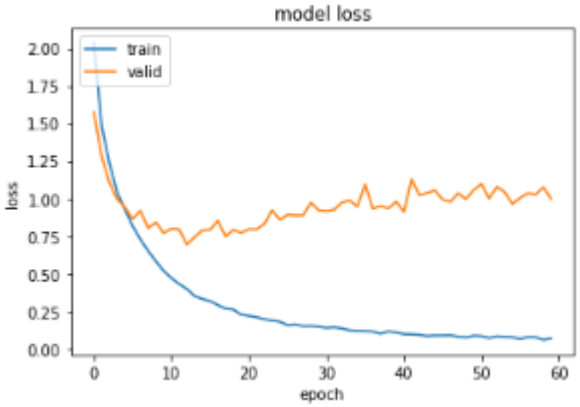

```
model.evaluate(valid_x,valid_y)
```

```
272/272 [==============================] -
[1.1566314381711624, 0.78308823529411764]
```

EXPERIMENT 7

Overview:	**Observation**:
• Defaults settings • Learning rate = 0.001 • **Batch –size = 500** • epochs=60 • optimizer = ADAM	• overfitting is observed • Test accuracy is approx. 81 % **Analysis**: Model still overfits






```
model.evaluate(valid_x,valid_y)
```

```
272/272 [==============================] -
[0.99709957838058472, 0.81617647058823528]
```

I'll be moving forward with this model (experiment 7)

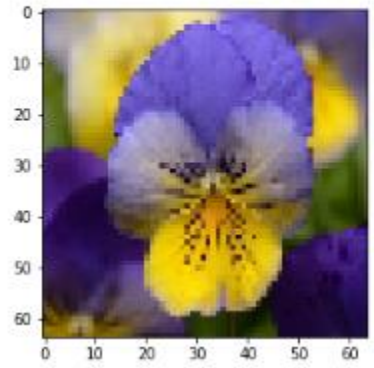
4.2 Model Evaluation and Validation

4.2.1Testing

To validate the solution model, we are testing it on few unseen images .

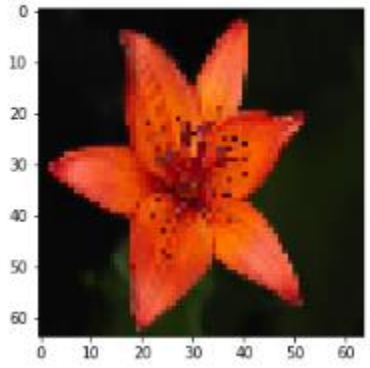
```
print(predict_flower(cv2.imread('test/pansy.jpg')))
```

```
Pansy
```



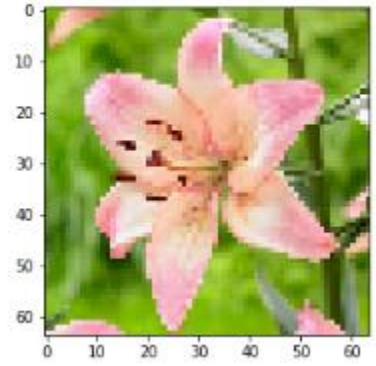
```
print(predict_flower(cv2.imread('test/tiger_lily.jpg')))
```

```
Tigerlily
```



```
print(predict_flower(cv2.imread('test/f1.jpg')))
```

```
Iris
```

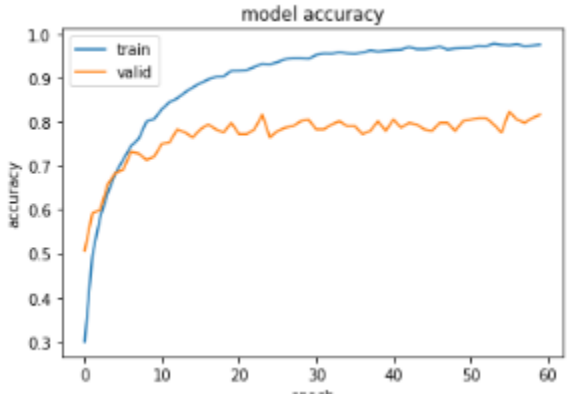


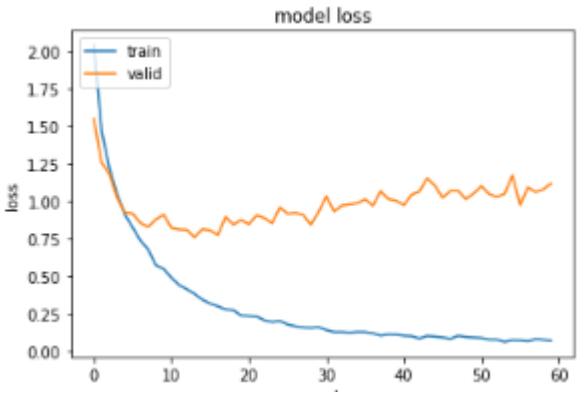
We can observe that our model is performing well on test data.

4.3 Solution Model Vs Benchmark Model

By performing a series of experiments I came to the conclusion that the model produced in Experiment 7 performs well on the given dataset.

The Accuracy was around 81 % and loss and accuracy plots are as below:



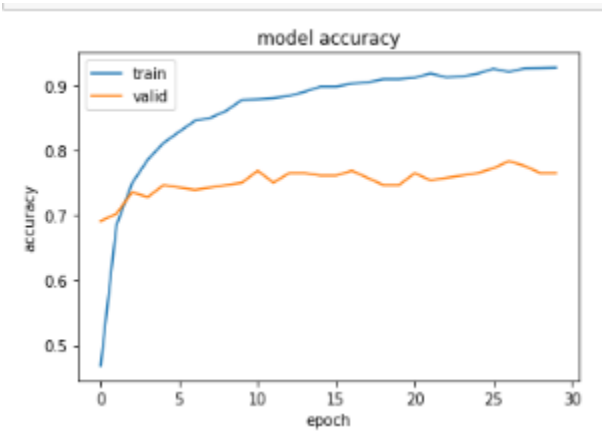


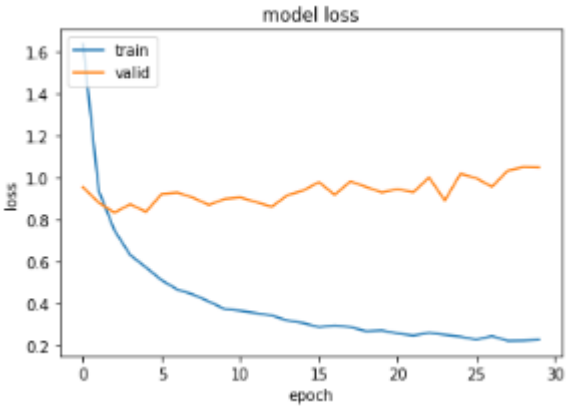

```
model.evaluate(valid_x,valid_y)
272/272 [==============================] -
[1.0478036482544506, 0.81985294117647056]
```

We can observe that the model still is overfitting and there is room for further improvements

I am only training the prediction head of VGG16 network implemented in keras with default parameters.

The performance of the Benchmark model is as follows:



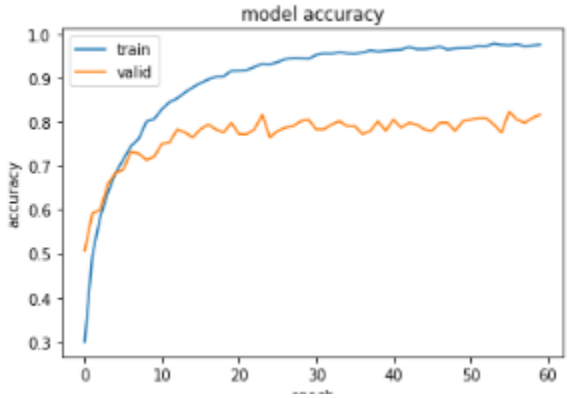


```
model.evaluate(valid_x,valid_y)
272/272 [==============================] -
[1.0504527442595537, 0.76470588235294112]
```

We can see from this comparison that our model is good choice for this problem statement.

5. Conclusion

5.1 Visualization



We can see that model is still overfitting .

5.2 Summary

A CNN works by extracting features from images. This eliminates the need for manual feature extraction. The features are not trained! They're learned while the network trains on a set of images. This makes deep learning models extremely accurate for computer vision tasks. CNNs learn feature detection through tens or hundreds of hidden layers. Each layer increases the complexity of the learned features.

5.2.1 Challenges

While testing the model I used opencv's imread() to read the image which uses BGR channel but while training I used keras's img_load() which uses RGB as channel, it took a while to figure out this was the reason the solution model was not performing well.

```
def predict_flower(img):
    img=img[...,::-1]
    img=cv2.resize(img, (64, 64))
    plt.imshow(img)
    x = np.expand_dims(img, axis=0)/255.
    prediction=model.predict(x)
    #print(prediction)
    return labels[np.argmax(prediction)]
```

So I reversed the channel after reading the image.

6 Future Scope

The solution model is overfitting to some extent.We might get better performance if we:

- train the model for longer duration(increase the number of epochs)
- tune the hyperparameters such as learning rate and batch size
- tune the parameters for data argumentation
- modify the network architecture

.References

- https://medium.com/
- https://stackoverflow.com/
- http://www.robots.ox.ac.uk/~vgg/data/flowers/17/
- https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html