## Import libraries and data

```
import tensorflow as tf
tf.__version__
```

```
'2.7.0'
```

```
# Tesorflow
import tensorflow.keras as keras
from tensorflow.keras.datasets import mnist
from tensorflow.keras.layers import Dense, Input, Flatten,\
                                    Reshape, LeakyReLU as LR,\
                                    Activation, Dropout
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D,
GlobalMaxPooling2D
from tensorflow.keras.callbacks import ReduceLROnPlateau,
ModelCheckpoint
from tensorflow.keras.layers import LeakyReLU
from keras.models import Sequential
from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D
from tensorflow.keras import layers
from tensorflow.keras.preprocessing import image as keras_image

# General
import warnings
warnings.filterwarnings('ignore')
from matplotlib import pyplot as plt
from IPython import display # If using IPython, Colab or Jupyter
import numpy as np
from sklearn.metrics import confusion_matrix
import seaborn as sns
from sklearn.metrics import accuracy_score, precision_score,
recall_score
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from tensorflow.keras import layers, losses

import pickle
import glob
import cv2

# Read images from folder
CLASS_NAMES = ['daisy', 'dandelion', 'roses', 'sunflowers', 'tulips']
# le.classes_
folders = ['daisy', 'dandelion', 'roses', 'sunflowers', 'tulips']

X_init = []
```

```
y_init = []

for folder in folders:
    path = 'data\\' + folder + '\\*.jpg'
    files = glob.glob(path)

    for file in files:
        img = cv2.imread(file)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        X_init.append(img)
        y_init.append(folder)
print('Count of images:', len(X_init))
```

```
Count of images: 2251
```

```
# Choose random 5 images and plot them
def plot_sample(X, y, is_gray = False):
    fig, axs = plt.subplots(1, 5, figsize=(20, 4))
    idxs = np.random.randint(0, len(X), size=5)

    for i, idx in enumerate(idxs):
        img = X[idx]
        ax = axs[i]
        if is_gray:
            ax.imshow(img, cmap='gray')
        else:
            ax.imshow(img)
        label = ""
        if isinstance(y, list):
            label = y[idx]
        else:
            label = CLASS_NAMES[y[idx].argmax()]
        ax.set_title(label)
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)
plot_sample(X_init, y_init)
```


## Prepocessing data and splitting dataset

Main steps in this section are:

- Convert a label column into several columns using one-hot encoder

- Resize and normalize images
- Augment images by rotation, shearing, flipping

```
# Convert label column into 5 one hot columns
le = preprocessing.LabelEncoder()
y_encoded = le.fit_transform(y_init).reshape(-1, 1)
enc = preprocessing.OneHotEncoder(handle_unknown='ignore')
y_one_hot = enc.fit_transform(y_encoded)
y = np.array(y_one_hot.toarray())

DIM = (128, 128)
# Resize and normalize images
def preprocess(X_init):
    X = []
    for i in range(len(X_init)):
        img = X_init[i].copy()
        img = cv2.resize(img, DIM, interpolation = cv2.INTER_AREA)
        #img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
        img = img / 255.0
        img = img.astype('float32')
        X.append(img)
    return X

X = preprocess(X_init)
X = np.array(X)
plot_sample(X, y)
```


```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.1, random_state=0) # split data
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train,
test_size=0.1, random_state=0) # split data
print("Train size:", len(X_train))
print("Val size:", len(X_val))
print("Test size:", len(X_test))
```

```
Train size: 1822
Val size: 203
Test size: 226
```

```
data_generator = keras_image.ImageDataGenerator(shear_range=0.3,
                                                zoom_range=0.3,
                                                rotation_range=30,
                                                horizontal_flip=True)
aug_iter = data_generator.flow(X_train, y_train)
```

```
X_train_aug = []
y_train_aug = []
for i in range(len(aug_iter)):
    it = next(aug_iter)
    for idx in range(it[0].shape[0]):
        X_train_aug.append(it[0][idx])
        y_train_aug.append(it[1][idx])
X_train_aug = np.array(X_train_aug)
y_train_aug = np.array(y_train_aug)
print(X_train_aug.shape)
print(y_train_aug.shape)
```

```
(1822, 128, 128, 3)
(1822, 5)
```

```
plot_sample(X_train_aug, y_train_aug)
```


# Models

## Utilities & Consts

```
BATCH_SIZE = 64
EPOCHS = 20
CLASS_NUM = 5
CLASS_NAMES = ['daisy', 'dandelion', 'roses', 'sunflowers', 'tulips']

def show_confusion_matrix(y_test, y_pred, in_prob = True, labels = [],
ax = None):
    # y_test & y_pred should be in the same format
    # in_prob == False implies that we get something like [2, 2, 3, 0,
...] where 2, 2... are numbers of class
    # in_prob = True implies that models might give probabilities
therefore we need to extract the most possible class
    if(in_prob):
        y_pred = np.argmax(y_pred, axis=1).T
        y_test = np.argmax(y_test, axis=1).T

    cf_matrix = confusion_matrix(y_test, y_pred)
    if(ax is None):
        fig, ax = plt.subplots(1, 1, figsize=(16, 4))

    sns.heatmap(cf_matrix, annot=True, cmap='Blues', fmt='.2f',
square=True, ax=ax)
```

```
    if len(labels) != 0:
        ax.xaxis.set_ticklabels(labels)
        ax.yaxis.set_ticklabels(labels)
    ax.set_title("Confusion matrix")
    ax.set_ylabel("True label")
    ax.set_xlabel("Predicted label")

# Example
#show_confusion_matrix(y_test, y_pred, True, labels = CLASS_NAMES)

def train_save_evaluate(model_func, path, X_train, y_train, X_val,
y_val, X_test, y_test, show_summary = True):
    model = model_func()
    if show_summary:
        model.summary()

    # train
    history = model.fit(X_train, y_train, epochs=EPOCHS,
batch_size=BATCH_SIZE, validation_data=(X_val, y_val))

    # evaluate
    fig, axs = plt.subplots(1, 3, figsize=(20, 4))
    ax = axs[0]
    ax.plot(history.history['accuracy'])
    ax.plot(history.history['val_accuracy'])
    ax.set_title('Model accuracy')
    ax.set_ylabel('Accuracy')
    ax.set_xlabel('Epoch')
    ax.legend(['Train', 'Val'], loc='upper left')

    ax = axs[1]
    ax.plot(history.history['loss'])
    ax.plot(history.history['val_loss'])
    ax.set_title('Model loss')
    ax.set_ylabel('Accuracy')
    ax.set_xlabel('Epoch')
    ax.legend(['Train', 'Val'], loc='upper left')

    ax = axs[2]
    y_pred = model.predict(X_test)
    show_confusion_matrix(y_test, y_pred, True, labels = CLASS_NAMES,
ax=ax)

    eval = model.evaluate(X_test, y_test)
    print("Loss: " + str(eval[0]) + " Accuracy: " + str(eval[1]))
    y_pred = np.argmax(y_pred, axis=1).T
    y_test2 = np.argmax(y_test, axis=1).T
    print(classification_report(y_test2, y_pred,
target_names=CLASS_NAMES))
```

```python
    # save
    model.save(path)
    model = tf.keras.models.load_model(path)

    return model

# inference mehod for CNN models
def inference_cnn(model, imgs):
    pred = []
    dim = (128, 128)
    for img in imgs:
        img = cv2.resize(img, dim, interpolation = cv2.INTER_AREA)
        img = img / 255
        img = np.expand_dims(img, axis=0)
        pred.append(model.predict(img))
    pred = np.array(pred)
    pred = pred.reshape(len(imgs), 5)
    return pred

# Example of using inference method
def plot_inference(inference_func, model, imgs, y_test, cols = 5,
with_prob = False):
    y_pred_prob = inference_func(model, imgs)
    y_pred = np.argmax(y_pred_prob, axis=1).T
    #y_test = np.argmax(y_test, axis=1).T
    names = np.array(CLASS_NAMES)
    n = len(imgs)
    k = 2 if with_prob else 1
    rows = int(np.ceil(n / cols))
    fig, axs = plt.subplots(k * rows, cols, figsize=(30, 6 * k *
rows))
    for i, img in enumerate(imgs):
        if with_prob:
            ax = axs[k * (i // cols) + 1, i % cols]
            prob = y_pred_prob[i]
            ind = np.argsort(prob)
            hbars = ax.barh(names[ind], prob[ind], color='#86bf91')
            ax.bar_label(hbars, fmt='%.2f')
            ax.set_xlabel('Probability')
            ax.set_xticks([])
            ax.set_xlim([0, 1.1])
            ax.spines['right'].set_visible(False)
            ax.spines['top'].set_visible(False)
            ax.spines['left'].set_visible(False)
            ax.spines['bottom'].set_visible(False)

        ax = axs[k * (i // cols), i % cols]
        #img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        ax.imshow(img)
```

```
        s = "Actual: " + y_test[i] + "\nPredicted: " +
CLASS_NAMES[y_pred[i]]
        ax.set_title(s)

        # borders
        color = 'red'
        if y_test[i] == CLASS_NAMES[y_pred[i]]:
            color = 'limegreen'

        ax.spines['bottom'].set_color(color)
        ax.spines['bottom'].set_linewidth(6)
        ax.spines['top'].set_color(color)
        ax.spines['top'].set_linewidth(6)
        ax.spines['right'].set_color(color)
        ax.spines['right'].set_linewidth(6)
        ax.spines['left'].set_color(color)
        ax.spines['left'].set_linewidth(6)

        # remove ticks
        ax.set_xticks([])
        ax.set_yticks([])
```

## CNN

This section includes building CNN models with different architecture with augmented data and with unaugmented data. For each model we calculated:

- plot with train and validation accuracies
- plot with train and validation losses to check whether the model is not overfitted
- confusion matrix
- precision, recall... for each classes

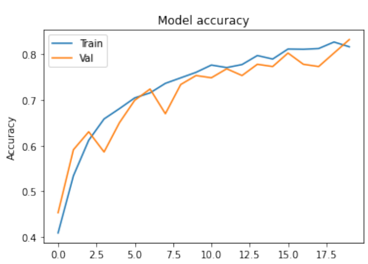
```
def model_1():
    model = Sequential()
    model.add(Conv2D(128, (3, 3), input_shape=X_train.shape[1:]))
    model.add(LeakyReLU(alpha=0.02))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    model.add(Conv2D(128, (3, 3)))
    model.add(LeakyReLU(alpha=0.02))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    model.add(GlobalMaxPooling2D())

    model.add(Dense(512))
    model.add(LeakyReLU(alpha=0.02))
    model.add(Dropout(0.5))
```

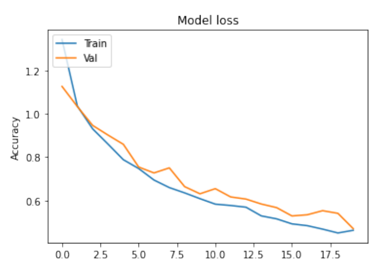
```
    model.add(Dense(CLASS_NUM))
    model.add(Activation('softmax'))

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

    return model

model = train_save_evaluate(model_1, 'models\CNN_01_aug.h5',
X_train_aug, y_train_aug, X_val, y_val, X_test, y_test)
```

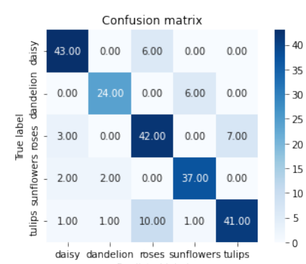
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 126, 126, 128)     3584

 leaky_re_lu (LeakyReLU)     (None, 126, 126, 128)     0

 max_pooling2d (MaxPooling2D  (None, 63, 63, 128)      0
 )

 dropout (Dropout)           (None, 63, 63, 128)       0

 conv2d_1 (Conv2D)           (None, 61, 61, 128)       147584

 leaky_re_lu_1 (LeakyReLU)   (None, 61, 61, 128)       0

 max_pooling2d_1 (MaxPooling  (None, 30, 30, 128)      0
 2D)

 dropout_1 (Dropout)         (None, 30, 30, 128)       0

 global_max_pooling2d (Globa  (None, 128)              0
 lMaxPooling2D)

 dense (Dense)               (None, 512)               66048

 leaky_re_lu_2 (LeakyReLU)   (None, 512)               0

 dropout_2 (Dropout)         (None, 512)               0

 dense_1 (Dense)             (None, 5)                 2565

 activation (Activation)     (None, 5)                 0

=================================================================
Total params: 219,781
Trainable params: 219,781
Non-trainable params: 0
_________________________________________________________________
```

```
Epoch 1/20
29/29 [==============================] - 129s 4s/step - loss: 1.3427 -
accuracy: 0.4089 - val_loss: 1.1264 - val_accuracy: 0.4532
Epoch 2/20
29/29 [==============================] - 133s 5s/step - loss: 1.0356 -
accuracy: 0.5340 - val_loss: 1.0333 - val_accuracy: 0.5911
Epoch 3/20
29/29 [==============================] - 154s 5s/step - loss: 0.9305 -
accuracy: 0.6120 - val_loss: 0.9463 - val_accuracy: 0.6305
Epoch 4/20
29/29 [==============================] - 149s 5s/step - loss: 0.8602 -
accuracy: 0.6586 - val_loss: 0.9018 - val_accuracy: 0.5862
Epoch 5/20
29/29 [==============================] - 151s 5s/step - loss: 0.7883 -
accuracy: 0.6811 - val_loss: 0.8592 - val_accuracy: 0.6502
Epoch 6/20
29/29 [==============================] - 164s 6s/step - loss: 0.7474 -
accuracy: 0.7047 - val_loss: 0.7545 - val_accuracy: 0.6995
Epoch 7/20
29/29 [==============================] - 125s 4s/step - loss: 0.6944 -
accuracy: 0.7157 - val_loss: 0.7271 - val_accuracy: 0.7241
Epoch 8/20
29/29 [==============================] - 110s 4s/step - loss: 0.6595 -
accuracy: 0.7366 - val_loss: 0.7506 - val_accuracy: 0.6700
Epoch 9/20
29/29 [==============================] - 109s 4s/step - loss: 0.6350 -
accuracy: 0.7486 - val_loss: 0.6638 - val_accuracy: 0.7340
Epoch 10/20
29/29 [==============================] - 120s 4s/step - loss: 0.6083 -
accuracy: 0.7607 - val_loss: 0.6309 - val_accuracy: 0.7537
Epoch 11/20
29/29 [==============================] - 146s 5s/step - loss: 0.5831 -
accuracy: 0.7766 - val_loss: 0.6545 - val_accuracy: 0.7488
Epoch 12/20
29/29 [==============================] - 141s 5s/step - loss: 0.5770 -
accuracy: 0.7711 - val_loss: 0.6163 - val_accuracy: 0.7685
Epoch 13/20
29/29 [==============================] - 151s 5s/step - loss: 0.5691 -
accuracy: 0.7777 - val_loss: 0.6065 - val_accuracy: 0.7537
Epoch 14/20
29/29 [==============================] - 135s 5s/step - loss: 0.5290 -
accuracy: 0.7975 - val_loss: 0.5838 - val_accuracy: 0.7783
Epoch 15/20
29/29 [==============================] - 158s 5s/step - loss: 0.5152 -
accuracy: 0.7898 - val_loss: 0.5671 - val_accuracy: 0.7734
Epoch 16/20
29/29 [==============================] - 132s 5s/step - loss: 0.4921 -
accuracy: 0.8117 - val_loss: 0.5287 - val_accuracy: 0.8030
Epoch 17/20
29/29 [==============================] - 128s 4s/step - loss: 0.4838 -
```

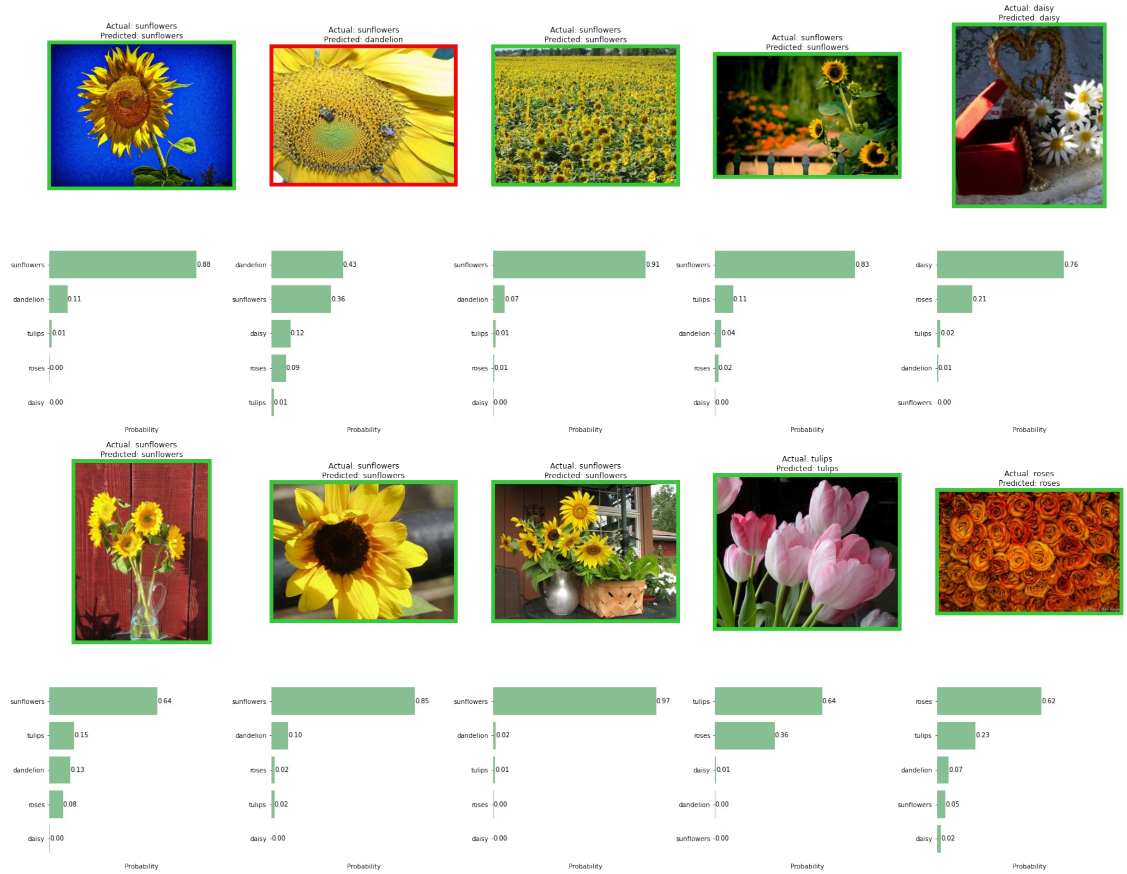
```
accuracy: 0.8112 - val_loss: 0.5341 - val_accuracy: 0.7783
Epoch 18/20
29/29 [==============================] - 115s 4s/step - loss: 0.4679 -
accuracy: 0.8128 - val_loss: 0.5530 - val_accuracy: 0.7734
Epoch 19/20
29/29 [==============================] - 117s 4s/step - loss: 0.4502 -
accuracy: 0.8271 - val_loss: 0.5407 - val_accuracy: 0.8030
Epoch 20/20
29/29 [==============================] - 133s 5s/step - loss: 0.4626 -
accuracy: 0.8167 - val_loss: 0.4699 - val_accuracy: 0.8325
8/8 [==============================] - 5s 593ms/step - loss: 0.4494 -
accuracy: 0.8274
Loss: 0.44937947392463684 Accuracy: 0.8274336457252502
              precision    recall  f1-score   support

       daisy       0.88      0.88      0.88        49
   dandelion       0.89      0.80      0.84        30
       roses       0.72      0.81      0.76        52
  sunflowers       0.84      0.90      0.87        41
      tulips       0.85      0.76      0.80        54

    accuracy                           0.83       226
   macro avg       0.84      0.83      0.83       226
weighted avg       0.83      0.83      0.83       226
```




```
idxs = np.random.randint(0, len(X_init), size=10)
imgs = [X_init[i] for i in idxs]
y_test_imgs = [y_init[i] for i in idxs]
plot_inference(inference_cnn, model, imgs, y_test_imgs, with_prob =
True)
```

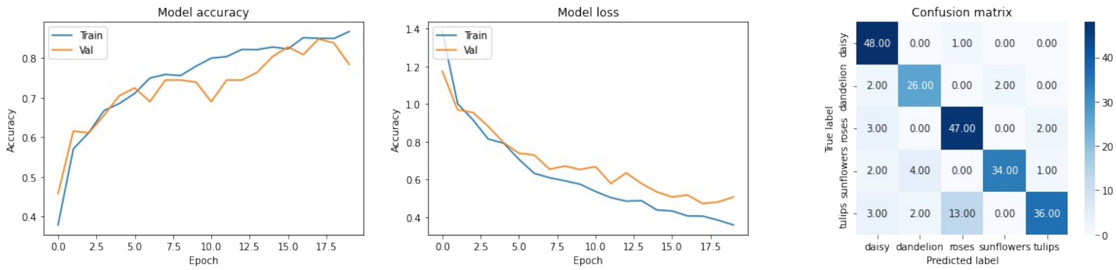



```
model = train_save_evaluate(model_1, 'models\CNN_01.h5', X_train,
y_train, X_val, y_val, X_test, y_test)
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_2 (Conv2D)           (None, 126, 126, 128)     3584

 leaky_re_lu_3 (LeakyReLU)   (None, 126, 126, 128)     0

 max_pooling2d_2 (MaxPooling  (None, 63, 63, 128)      0
 2D)

 dropout_3 (Dropout)         (None, 63, 63, 128)       0

 conv2d_3 (Conv2D)           (None, 61, 61, 128)       147584

 leaky_re_lu_4 (LeakyReLU)   (None, 61, 61, 128)       0

 max_pooling2d_3 (MaxPooling  (None, 30, 30, 128)      0
 2D)

 dropout_4 (Dropout)         (None, 30, 30, 128)       0
```

```
 global_max_pooling2d_1 (Glo  (None, 128)              0
 balMaxPooling2D)

 dense_2 (Dense)             (None, 512)               66048

 leaky_re_lu_5 (LeakyReLU)   (None, 512)               0

 dropout_5 (Dropout)         (None, 512)               0

 dense_3 (Dense)             (None, 5)                 2565

 activation_1 (Activation)   (None, 5)                 0

=================================================================
Total params: 219,781
Trainable params: 219,781
Non-trainable params: 0
_________________________________________________________________
Epoch 1/20
29/29 [==============================] - 152s 5s/step - loss: 1.3838 -
accuracy: 0.3793 - val_loss: 1.1732 - val_accuracy: 0.4581
Epoch 2/20
29/29 [==============================] - 142s 5s/step - loss: 1.0016 -
accuracy: 0.5708 - val_loss: 0.9692 - val_accuracy: 0.6158
Epoch 3/20
29/29 [==============================] - 163s 6s/step - loss: 0.9154 -
accuracy: 0.6114 - val_loss: 0.9547 - val_accuracy: 0.6108
Epoch 4/20
29/29 [==============================] - 116s 4s/step - loss: 0.8136 -
accuracy: 0.6668 - val_loss: 0.8810 - val_accuracy: 0.6552
Epoch 5/20
29/29 [==============================] - 111s 4s/step - loss: 0.7923 -
accuracy: 0.6850 - val_loss: 0.7947 - val_accuracy: 0.7044
Epoch 6/20
29/29 [==============================] - 149s 5s/step - loss: 0.7067 -
accuracy: 0.7102 - val_loss: 0.7387 - val_accuracy: 0.7241
Epoch 7/20
29/29 [==============================] - 117s 4s/step - loss: 0.6322 -
accuracy: 0.7492 - val_loss: 0.7295 - val_accuracy: 0.6897
Epoch 8/20
29/29 [==============================] - 122s 4s/step - loss: 0.6091 -
accuracy: 0.7580 - val_loss: 0.6540 - val_accuracy: 0.7438
Epoch 9/20
29/29 [==============================] - 138s 5s/step - loss: 0.5925 -
accuracy: 0.7552 - val_loss: 0.6708 - val_accuracy: 0.7438
Epoch 10/20
29/29 [==============================] - 125s 4s/step - loss: 0.5744 -
accuracy: 0.7788 - val_loss: 0.6529 - val_accuracy: 0.7389
Epoch 11/20
```

```
29/29 [==============================] - 120s 4s/step - loss: 0.5360 -
accuracy: 0.7991 - val_loss: 0.6674 - val_accuracy: 0.6897
Epoch 12/20
29/29 [==============================] - 110s 4s/step - loss: 0.5036 -
accuracy: 0.8030 - val_loss: 0.5782 - val_accuracy: 0.7438
Epoch 13/20
29/29 [==============================] - 111s 4s/step - loss: 0.4848 -
accuracy: 0.8211 - val_loss: 0.6348 - val_accuracy: 0.7438
Epoch 14/20
29/29 [==============================] - 105s 4s/step - loss: 0.4874 -
accuracy: 0.8205 - val_loss: 0.5783 - val_accuracy: 0.7635
Epoch 15/20
29/29 [==============================] - 106s 4s/step - loss: 0.4383 -
accuracy: 0.8271 - val_loss: 0.5338 - val_accuracy: 0.8030
Epoch 16/20
29/29 [==============================] - 109s 4s/step - loss: 0.4327 -
accuracy: 0.8222 - val_loss: 0.5068 - val_accuracy: 0.8276
Epoch 17/20
29/29 [==============================] - 107s 4s/step - loss: 0.4067 -
accuracy: 0.8507 - val_loss: 0.5178 - val_accuracy: 0.8079
Epoch 18/20
29/29 [==============================] - 106s 4s/step - loss: 0.4056 -
accuracy: 0.8491 - val_loss: 0.4718 - val_accuracy: 0.8473
Epoch 19/20
29/29 [==============================] - 108s 4s/step - loss: 0.3841 -
accuracy: 0.8485 - val_loss: 0.4806 - val_accuracy: 0.8374
Epoch 20/20
29/29 [==============================] - 111s 4s/step - loss: 0.3587 -
accuracy: 0.8661 - val_loss: 0.5064 - val_accuracy: 0.7833
8/8 [==============================] - 3s 392ms/step - loss: 0.4239 -
accuracy: 0.8451
Loss: 0.4238995313644409 Accuracy: 0.8451327681541443
              precision    recall  f1-score   support

       daisy       0.83      0.98      0.90        49
   dandelion       0.81      0.87      0.84        30
       roses       0.77      0.90      0.83        52
  sunflowers       0.94      0.83      0.88        41
      tulips       0.92      0.67      0.77        54

    accuracy                           0.85       226
   macro avg       0.86      0.85      0.85       226
weighted avg       0.86      0.85      0.84       226
```

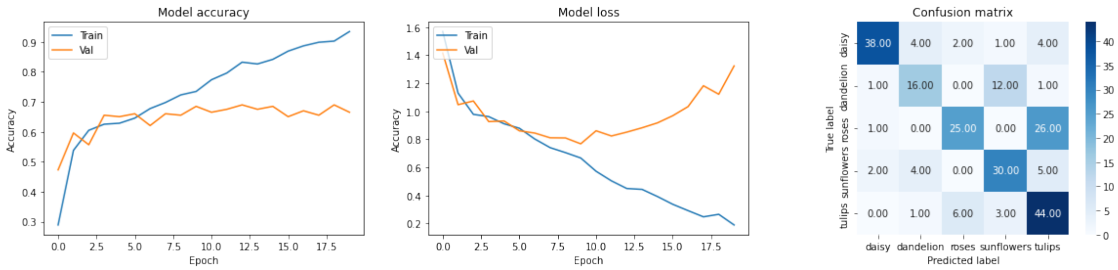



```
def model_2():
    model = Sequential()
    model.add(Conv2D(32, (5, 5), activation='relu',
input_shape=X_train.shape[1:]))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.4))

    model.add(Conv2D(64, (5, 5), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.4))

    model.add(Conv2D(64, (5, 5), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.4))

    model.add(Flatten())
    model.add(Dense(64, activation='relu'))
    model.add(Dense(CLASS_NUM,  activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])

    return model

model = train_save_evaluate(model_2, 'models\CNN_02_aug.h5',
X_train_aug, y_train_aug, X_val, y_val, X_test, y_test)
```

```
Model: "sequential_2"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 124, 124, 32)      2432

 max_pooling2d_4 (MaxPooling  (None, 62, 62, 32)       0
 2D)

 dropout_6 (Dropout)         (None, 62, 62, 32)        0

 conv2d_5 (Conv2D)           (None, 58, 58, 64)        51264

 max_pooling2d_5 (MaxPooling  (None, 29, 29, 64)       0
 2D)
```

```
 dropout_7 (Dropout)         (None, 29, 29, 64)        0

 conv2d_6 (Conv2D)           (None, 25, 25, 64)        102464

 max_pooling2d_6 (MaxPooling  (None, 12, 12, 64)       0
 2D)

 dropout_8 (Dropout)         (None, 12, 12, 64)        0

 flatten (Flatten)           (None, 9216)              0

 dense_4 (Dense)             (None, 64)                589888

 dense_5 (Dense)             (None, 5)                 325

=================================================================
Total params: 746,373
Trainable params: 746,373
Non-trainable params: 0
_________________________________________________________________
Epoch 1/20
29/29 [==============================] - 48s 2s/step - loss: 1.5688 -
accuracy: 0.2892 - val_loss: 1.4153 - val_accuracy: 0.4729
Epoch 2/20
29/29 [==============================] - 46s 2s/step - loss: 1.1297 -
accuracy: 0.5379 - val_loss: 1.0471 - val_accuracy: 0.5961
Epoch 3/20
29/29 [==============================] - 47s 2s/step - loss: 0.9781 -
accuracy: 0.6048 - val_loss: 1.0735 - val_accuracy: 0.5567
Epoch 4/20
29/29 [==============================] - 46s 2s/step - loss: 0.9620 -
accuracy: 0.6251 - val_loss: 0.9264 - val_accuracy: 0.6552
Epoch 5/20
29/29 [==============================] - 46s 2s/step - loss: 0.9109 -
accuracy: 0.6284 - val_loss: 0.9307 - val_accuracy: 0.6502
Epoch 6/20
29/29 [==============================] - 46s 2s/step - loss: 0.8790 -
accuracy: 0.6454 - val_loss: 0.8608 - val_accuracy: 0.6601
Epoch 7/20
29/29 [==============================] - 46s 2s/step - loss: 0.8023 -
accuracy: 0.6773 - val_loss: 0.8448 - val_accuracy: 0.6207
Epoch 8/20
29/29 [==============================] - 46s 2s/step - loss: 0.7409 -
accuracy: 0.6976 - val_loss: 0.8111 - val_accuracy: 0.6601
Epoch 9/20
29/29 [==============================] - 47s 2s/step - loss: 0.7053 -
accuracy: 0.7228 - val_loss: 0.8098 - val_accuracy: 0.6552
Epoch 10/20
29/29 [==============================] - 46s 2s/step - loss: 0.6668 -
accuracy: 0.7344 - val_loss: 0.7679 - val_accuracy: 0.6847
```

```
Epoch 11/20
29/29 [==============================] - 46s 2s/step - loss: 0.5722 -
accuracy: 0.7733 - val_loss: 0.8611 - val_accuracy: 0.6650
Epoch 12/20
29/29 [==============================] - 46s 2s/step - loss: 0.5038 -
accuracy: 0.7958 - val_loss: 0.8239 - val_accuracy: 0.6749
Epoch 13/20
29/29 [==============================] - 46s 2s/step - loss: 0.4491 -
accuracy: 0.8321 - val_loss: 0.8515 - val_accuracy: 0.6897
Epoch 14/20
29/29 [==============================] - 46s 2s/step - loss: 0.4428 -
accuracy: 0.8260 - val_loss: 0.8824 - val_accuracy: 0.6749
Epoch 15/20
29/29 [==============================] - 46s 2s/step - loss: 0.3921 -
accuracy: 0.8414 - val_loss: 0.9172 - val_accuracy: 0.6847
Epoch 16/20
29/29 [==============================] - 47s 2s/step - loss: 0.3366 -
accuracy: 0.8688 - val_loss: 0.9685 - val_accuracy: 0.6502
Epoch 17/20
29/29 [==============================] - 46s 2s/step - loss: 0.2913 -
accuracy: 0.8864 - val_loss: 1.0335 - val_accuracy: 0.6700
Epoch 18/20
29/29 [==============================] - 47s 2s/step - loss: 0.2471 -
accuracy: 0.8985 - val_loss: 1.1823 - val_accuracy: 0.6552
Epoch 19/20
29/29 [==============================] - 47s 2s/step - loss: 0.2651 -
accuracy: 0.9023 - val_loss: 1.1215 - val_accuracy: 0.6897
Epoch 20/20
29/29 [==============================] - 46s 2s/step - loss: 0.1892 -
accuracy: 0.9341 - val_loss: 1.3235 - val_accuracy: 0.6650
8/8 [==============================] - 1s 149ms/step - loss: 1.0754 -
accuracy: 0.6770
Loss: 1.0754141807556152 Accuracy: 0.6769911646842957
              precision    recall  f1-score   support

       daisy       0.90      0.78      0.84        49
   dandelion       0.64      0.53      0.58        30
       roses       0.76      0.48      0.59        52
  sunflowers       0.65      0.73      0.69        41
      tulips       0.55      0.81      0.66        54

    accuracy                           0.68       226
   macro avg       0.70      0.67      0.67       226
weighted avg       0.71      0.68      0.68       226
```

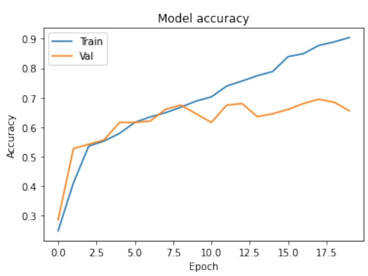


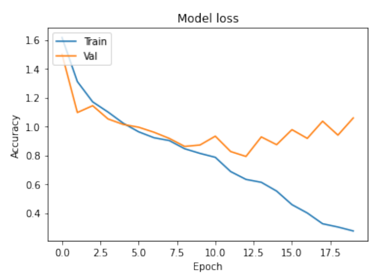

```
model = train_save_evaluate(model_2, 'models\CNN_02.h5', X_train,
y_train, X_val, y_val, X_test, y_test)
```

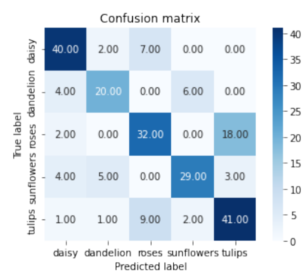
```
Model: "sequential_3"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_7 (Conv2D)           (None, 124, 124, 32)      2432

 max_pooling2d_7 (MaxPooling  (None, 62, 62, 32)       0
 2D)

 dropout_9 (Dropout)         (None, 62, 62, 32)        0

 conv2d_8 (Conv2D)           (None, 58, 58, 64)        51264

 max_pooling2d_8 (MaxPooling  (None, 29, 29, 64)       0
 2D)

 dropout_10 (Dropout)        (None, 29, 29, 64)        0

 conv2d_9 (Conv2D)           (None, 25, 25, 64)        102464

 max_pooling2d_9 (MaxPooling  (None, 12, 12, 64)       0
 2D)

 dropout_11 (Dropout)        (None, 12, 12, 64)        0

 flatten_1 (Flatten)         (None, 9216)              0

 dense_6 (Dense)             (None, 64)                589888

 dense_7 (Dense)             (None, 5)                 325

=================================================================
Total params: 746,373
Trainable params: 746,373
Non-trainable params: 0
_________________________________________________________________
Epoch 1/20
29/29 [==============================] - 46s 2s/step - loss: 1.6155 -
accuracy: 0.2475 - val_loss: 1.5000 - val_accuracy: 0.2857
```

```
Epoch 2/20
29/29 [==============================] - 46s 2s/step - loss: 1.3105 -
accuracy: 0.4105 - val_loss: 1.0962 - val_accuracy: 0.5271
Epoch 3/20
29/29 [==============================] - 46s 2s/step - loss: 1.1698 -
accuracy: 0.5357 - val_loss: 1.1435 - val_accuracy: 0.5419
Epoch 4/20
29/29 [==============================] - 46s 2s/step - loss: 1.0995 -
accuracy: 0.5527 - val_loss: 1.0518 - val_accuracy: 0.5567
Epoch 5/20
29/29 [==============================] - 45s 2s/step - loss: 1.0217 -
accuracy: 0.5785 - val_loss: 1.0134 - val_accuracy: 0.6158
Epoch 6/20
29/29 [==============================] - 46s 2s/step - loss: 0.9625 -
accuracy: 0.6158 - val_loss: 0.9943 - val_accuracy: 0.6158
Epoch 7/20
29/29 [==============================] - 46s 2s/step - loss: 0.9216 -
accuracy: 0.6345 - val_loss: 0.9595 - val_accuracy: 0.6207
Epoch 8/20
29/29 [==============================] - 46s 2s/step - loss: 0.9017 -
accuracy: 0.6487 - val_loss: 0.9168 - val_accuracy: 0.6601
Epoch 9/20
29/29 [==============================] - 47s 2s/step - loss: 0.8453 -
accuracy: 0.6674 - val_loss: 0.8614 - val_accuracy: 0.6749
Epoch 10/20
29/29 [==============================] - 46s 2s/step - loss: 0.8132 -
accuracy: 0.6883 - val_loss: 0.8711 - val_accuracy: 0.6453
Epoch 11/20
29/29 [==============================] - 46s 2s/step - loss: 0.7859 -
accuracy: 0.7025 - val_loss: 0.9324 - val_accuracy: 0.6158
Epoch 12/20
29/29 [==============================] - 45s 2s/step - loss: 0.6872 -
accuracy: 0.7393 - val_loss: 0.8259 - val_accuracy: 0.6749
Epoch 13/20
29/29 [==============================] - 46s 2s/step - loss: 0.6329 -
accuracy: 0.7563 - val_loss: 0.7919 - val_accuracy: 0.6798
Epoch 14/20
29/29 [==============================] - 45s 2s/step - loss: 0.6129 -
accuracy: 0.7744 - val_loss: 0.9271 - val_accuracy: 0.6355
Epoch 15/20
29/29 [==============================] - 46s 2s/step - loss: 0.5520 -
accuracy: 0.7887 - val_loss: 0.8734 - val_accuracy: 0.6453
Epoch 16/20
29/29 [==============================] - 45s 2s/step - loss: 0.4580 -
accuracy: 0.8392 - val_loss: 0.9776 - val_accuracy: 0.6601
Epoch 17/20
29/29 [==============================] - 45s 2s/step - loss: 0.4002 -
accuracy: 0.8491 - val_loss: 0.9170 - val_accuracy: 0.6798
Epoch 18/20
29/29 [==============================] - 46s 2s/step - loss: 0.3258 -
```

```
accuracy: 0.8771 - val_loss: 1.0359 - val_accuracy: 0.6946
Epoch 19/20
29/29 [==============================] - 45s 2s/step - loss: 0.3031 -
accuracy: 0.8891 - val_loss: 0.9396 - val_accuracy: 0.6847
Epoch 20/20
29/29 [==============================] - 46s 2s/step - loss: 0.2758 -
accuracy: 0.9040 - val_loss: 1.0587 - val_accuracy: 0.6552
8/8 [==============================] - 1s 154ms/step - loss: 0.9301 -
accuracy: 0.71680s - loss: 0.8533 - accuracy: 0.
Loss: 0.9301110506057739 Accuracy: 0.7168141603469849
              precision    recall  f1-score   support

       daisy       0.78      0.82      0.80        49
   dandelion       0.71      0.67      0.69        30
       roses       0.67      0.62      0.64        52
  sunflowers       0.78      0.71      0.74        41
      tulips       0.66      0.76      0.71        54

    accuracy                           0.72       226
   macro avg       0.72      0.71      0.72       226
weighted avg       0.72      0.72      0.72       226
```




## CNN using pretrained model

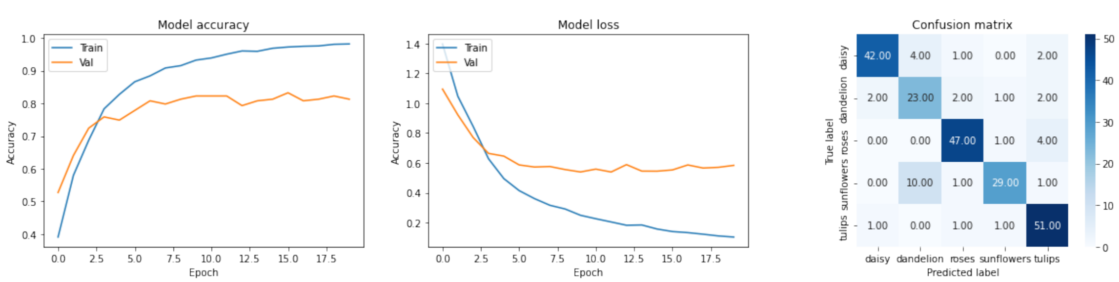
In this model we have built CNN using some pretrained models such as InceptionV3, DenseNet201.We use all layers except last one and froze their parameters. Then we have added several dense layers and train such models.

```
dim1 = X_train[0].shape[0] # image width
dim2 = X_train[0].shape[1] # image height
dim3 = 3 # image channels

from keras.applications.inception_v3 import InceptionV3
def model_3():
    model = InceptionV3(include_top=False, input_shape=(dim1, dim2,
dim3))
    model.trainable=False
    flat1 = tf.keras.layers.GlobalAveragePooling2D()(model.layers[-
1].output)
    class1 = Dense(10, activation='relu')(flat1)
    output = Dense(5, activation='softmax')(class1)
```

```
    model = Model(inputs=model.inputs, outputs=output)
    model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
    return model

model = train_save_evaluate(model_3, 'models\CNN_03_aug.h5',
X_train_aug, y_train_aug, X_val, y_val, X_test, y_test, show_summary =
False)
```

```
Epoch 1/20
29/29 [==============================] - 30s 875ms/step - loss: 1.3975
- accuracy: 0.3908 - val_loss: 1.0950 - val_accuracy: 0.5271
Epoch 2/20
29/29 [==============================] - 24s 847ms/step - loss: 1.0468
- accuracy: 0.5796 - val_loss: 0.9221 - val_accuracy: 0.6404
Epoch 3/20
29/29 [==============================] - 26s 883ms/step - loss: 0.8441
- accuracy: 0.6872 - val_loss: 0.7701 - val_accuracy: 0.7241
Epoch 4/20
29/29 [==============================] - 25s 863ms/step - loss: 0.6280
- accuracy: 0.7832 - val_loss: 0.6640 - val_accuracy: 0.7586
Epoch 5/20
29/29 [==============================] - 25s 856ms/step - loss: 0.4948
- accuracy: 0.8277 - val_loss: 0.6456 - val_accuracy: 0.7488
Epoch 6/20
29/29 [==============================] - 25s 854ms/step - loss: 0.4144
- accuracy: 0.8661 - val_loss: 0.5866 - val_accuracy: 0.7783
Epoch 7/20
29/29 [==============================] - 25s 864ms/step - loss: 0.3608
- accuracy: 0.8842 - val_loss: 0.5725 - val_accuracy: 0.8079
Epoch 8/20
29/29 [==============================] - 25s 854ms/step - loss: 0.3162
- accuracy: 0.9083 - val_loss: 0.5760 - val_accuracy: 0.7980
Epoch 9/20
29/29 [==============================] - 24s 848ms/step - loss: 0.2911
- accuracy: 0.9155 - val_loss: 0.5555 - val_accuracy: 0.8128
Epoch 10/20
29/29 [==============================] - 25s 853ms/step - loss: 0.2490
- accuracy: 0.9325 - val_loss: 0.5393 - val_accuracy: 0.8227
Epoch 11/20
29/29 [==============================] - 25s 850ms/step - loss: 0.2259
- accuracy: 0.9391 - val_loss: 0.5585 - val_accuracy: 0.8227
Epoch 12/20
29/29 [==============================] - 25s 870ms/step - loss: 0.2042
- accuracy: 0.9506 - val_loss: 0.5391 - val_accuracy: 0.8227
Epoch 13/20
29/29 [==============================] - 25s 859ms/step - loss: 0.1821
- accuracy: 0.9605 - val_loss: 0.5886 - val_accuracy: 0.7931
Epoch 14/20
29/29 [==============================] - 25s 876ms/step - loss: 0.1841
- accuracy: 0.9594 - val_loss: 0.5451 - val_accuracy: 0.8079
```

```
Epoch 15/20
29/29 [==============================] - 25s 859ms/step - loss: 0.1570
- accuracy: 0.9687 - val_loss: 0.5447 - val_accuracy: 0.8128
Epoch 16/20
29/29 [==============================] - 25s 871ms/step - loss: 0.1401
- accuracy: 0.9726 - val_loss: 0.5530 - val_accuracy: 0.8325
Epoch 17/20
29/29 [==============================] - 25s 858ms/step - loss: 0.1329
- accuracy: 0.9748 - val_loss: 0.5864 - val_accuracy: 0.8079
Epoch 18/20
29/29 [==============================] - 25s 850ms/step - loss: 0.1223
- accuracy: 0.9759 - val_loss: 0.5657 - val_accuracy: 0.8128
Epoch 19/20
29/29 [==============================] - 25s 854ms/step - loss: 0.1107
- accuracy: 0.9808 - val_loss: 0.5704 - val_accuracy: 0.8227
Epoch 20/20
29/29 [==============================] - 25s 849ms/step - loss: 0.1032
- accuracy: 0.9819 - val_loss: 0.5836 - val_accuracy: 0.8128
8/8 [==============================] - 3s 353ms/step - loss: 0.5233 -
accuracy: 0.8496
Loss: 0.5233474373817444 Accuracy: 0.8495575189590454
              precision    recall  f1-score   support

       daisy       0.93      0.86      0.89        49
   dandelion       0.62      0.77      0.69        30
       roses       0.90      0.90      0.90        52
  sunflowers       0.91      0.71      0.79        41
      tulips       0.85      0.94      0.89        54

    accuracy                           0.85       226
   macro avg       0.84      0.84      0.83       226
weighted avg       0.86      0.85      0.85       226
```

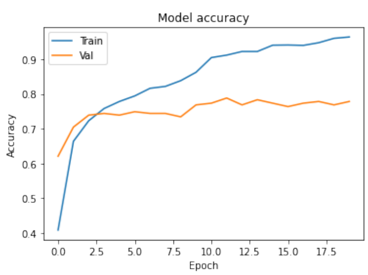


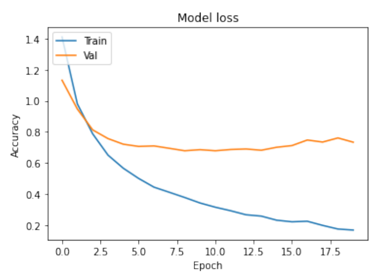

```python
model = train_save_evaluate(model_3, 'models\CNN_03.h5', X_train,
y_train, X_val, y_val, X_test, y_test, show_summary = False)
```

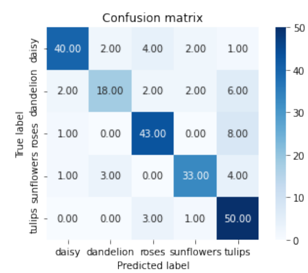
```
Epoch 1/20
29/29 [==============================] - 31s 902ms/step - loss: 1.4099
- accuracy: 0.4083 - val_loss: 1.1325 - val_accuracy: 0.6207
Epoch 2/20
29/29 [==============================] - 25s 850ms/step - loss: 0.9801
```

```
- accuracy: 0.6636 - val_loss: 0.9474 - val_accuracy: 0.7044
Epoch 3/20
29/29 [==============================] - 25s 864ms/step - loss: 0.7881
- accuracy: 0.7228 - val_loss: 0.8132 - val_accuracy: 0.7389
Epoch 4/20
29/29 [==============================] - 25s 864ms/step - loss: 0.6512
- accuracy: 0.7580 - val_loss: 0.7574 - val_accuracy: 0.7438
Epoch 5/20
29/29 [==============================] - 25s 871ms/step - loss: 0.5668
- accuracy: 0.7783 - val_loss: 0.7213 - val_accuracy: 0.7389
Epoch 6/20
29/29 [==============================] - 26s 899ms/step - loss: 0.5013
- accuracy: 0.7942 - val_loss: 0.7071 - val_accuracy: 0.7488
Epoch 7/20
29/29 [==============================] - 25s 864ms/step - loss: 0.4455
- accuracy: 0.8161 - val_loss: 0.7103 - val_accuracy: 0.7438
Epoch 8/20
29/29 [==============================] - 26s 886ms/step - loss: 0.4126
- accuracy: 0.8216 - val_loss: 0.6950 - val_accuracy: 0.7438
Epoch 9/20
29/29 [==============================] - 25s 852ms/step - loss: 0.3792
- accuracy: 0.8381 - val_loss: 0.6793 - val_accuracy: 0.7340
Epoch 10/20
29/29 [==============================] - 25s 883ms/step - loss: 0.3439
- accuracy: 0.8622 - val_loss: 0.6863 - val_accuracy: 0.7685
Epoch 11/20
29/29 [==============================] - 25s 875ms/step - loss: 0.3168
- accuracy: 0.9045 - val_loss: 0.6793 - val_accuracy: 0.7734
Epoch 12/20
29/29 [==============================] - 25s 864ms/step - loss: 0.2940
- accuracy: 0.9116 - val_loss: 0.6874 - val_accuracy: 0.7882
Epoch 13/20
29/29 [==============================] - 25s 871ms/step - loss: 0.2683
- accuracy: 0.9221 - val_loss: 0.6907 - val_accuracy: 0.7685
Epoch 14/20
29/29 [==============================] - 25s 859ms/step - loss: 0.2595
- accuracy: 0.9221 - val_loss: 0.6827 - val_accuracy: 0.7833
Epoch 15/20
29/29 [==============================] - 25s 878ms/step - loss: 0.2333
- accuracy: 0.9402 - val_loss: 0.7017 - val_accuracy: 0.7734
Epoch 16/20
29/29 [==============================] - 25s 872ms/step - loss: 0.2231
- accuracy: 0.9407 - val_loss: 0.7126 - val_accuracy: 0.7635
Epoch 17/20
29/29 [==============================] - 25s 869ms/step - loss: 0.2266
- accuracy: 0.9396 - val_loss: 0.7482 - val_accuracy: 0.7734
Epoch 18/20
29/29 [==============================] - 26s 893ms/step - loss: 0.2000
- accuracy: 0.9473 - val_loss: 0.7352 - val_accuracy: 0.7783
Epoch 19/20
```

```
29/29 [==============================] - 26s 891ms/step - loss: 0.1768
- accuracy: 0.9599 - val_loss: 0.7616 - val_accuracy: 0.7685
Epoch 20/20
29/29 [==============================] - 26s 910ms/step - loss: 0.1701
- accuracy: 0.9638 - val_loss: 0.7342 - val_accuracy: 0.7783
8/8 [==============================] - 3s 410ms/step - loss: 0.6442 -
accuracy: 0.8142
Loss: 0.6442194581031799 Accuracy: 0.8141592741012573
              precision    recall  f1-score   support

       daisy       0.91      0.82      0.86        49
   dandelion       0.78      0.60      0.68        30
       roses       0.83      0.83      0.83        52
  sunflowers       0.87      0.80      0.84        41
      tulips       0.72      0.93      0.81        54

    accuracy                           0.81       226
   macro avg       0.82      0.79      0.80       226
weighted avg       0.82      0.81      0.81       226
```



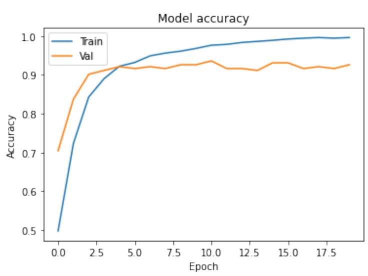



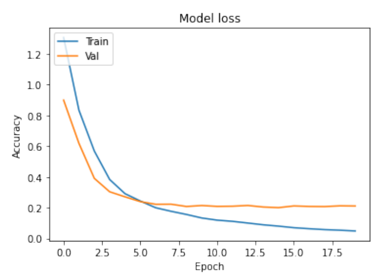

```
from keras.applications.densenet import DenseNet201
def model_4():
    model = DenseNet201(weights='imagenet', include_top=False,
input_shape=(dim1, dim2, dim3))
    #InceptionV3(include_top=False, input_shape=(dim1, dim2, dim3))
    model.trainable=False
    flat1 = tf.keras.layers.GlobalAveragePooling2D()(model.layers[-
1].output)
    class1 = Dense(10, activation='relu')(flat1)
    output = Dense(5, activation='softmax')(class1)
    model = Model(inputs=model.inputs, outputs=output)
    model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
    return model
model = train_save_evaluate(model_4, 'models\CNN_04_aug.h5',
X_train_aug, y_train_aug, X_val, y_val, X_test, y_test, show_summary =
False)
```

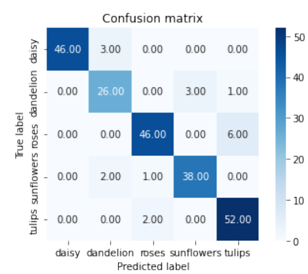
```
Epoch 1/20
29/29 [==============================] - 156s 5s/step - loss: 1.3072 -
accuracy: 0.4978 - val_loss: 0.9000 - val_accuracy: 0.7044
```

```
Epoch 2/20
29/29 [==============================] - 142s 5s/step - loss: 0.8339 -
accuracy: 0.7234 - val_loss: 0.6182 - val_accuracy: 0.8374
Epoch 3/20
29/29 [==============================] - 147s 5s/step - loss: 0.5687 -
accuracy: 0.8430 - val_loss: 0.3909 - val_accuracy: 0.9015
Epoch 4/20
29/29 [==============================] - 143s 5s/step - loss: 0.3834 -
accuracy: 0.8902 - val_loss: 0.3034 - val_accuracy: 0.9113
Epoch 5/20
29/29 [==============================] - 144s 5s/step - loss: 0.2910 -
accuracy: 0.9221 - val_loss: 0.2705 - val_accuracy: 0.9212
Epoch 6/20
29/29 [==============================] - 144s 5s/step - loss: 0.2432 -
accuracy: 0.9319 - val_loss: 0.2394 - val_accuracy: 0.9163
Epoch 7/20
29/29 [==============================] - 143s 5s/step - loss: 0.1995 -
accuracy: 0.9490 - val_loss: 0.2220 - val_accuracy: 0.9212
Epoch 8/20
29/29 [==============================] - 144s 5s/step - loss: 0.1764 -
accuracy: 0.9561 - val_loss: 0.2226 - val_accuracy: 0.9163
Epoch 9/20
29/29 [==============================] - 143s 5s/step - loss: 0.1559 -
accuracy: 0.9610 - val_loss: 0.2077 - val_accuracy: 0.9261
Epoch 10/20
29/29 [==============================] - 142s 5s/step - loss: 0.1329 -
accuracy: 0.9682 - val_loss: 0.2139 - val_accuracy: 0.9261
Epoch 11/20
29/29 [==============================] - 144s 5s/step - loss: 0.1188 -
accuracy: 0.9764 - val_loss: 0.2084 - val_accuracy: 0.9360
Epoch 12/20
29/29 [==============================] - 142s 5s/step - loss: 0.1111 -
accuracy: 0.9786 - val_loss: 0.2094 - val_accuracy: 0.9163
Epoch 13/20
29/29 [==============================] - 143s 5s/step - loss: 0.1000 -
accuracy: 0.9835 - val_loss: 0.2143 - val_accuracy: 0.9163
Epoch 14/20
29/29 [==============================] - 152s 5s/step - loss: 0.0887 -
accuracy: 0.9863 - val_loss: 0.2043 - val_accuracy: 0.9113
Epoch 15/20
29/29 [==============================] - 144s 5s/step - loss: 0.0799 -
accuracy: 0.9890 - val_loss: 0.2002 - val_accuracy: 0.9310
Epoch 16/20
29/29 [==============================] - 143s 5s/step - loss: 0.0698 -
accuracy: 0.9923 - val_loss: 0.2113 - val_accuracy: 0.9310
Epoch 17/20
29/29 [==============================] - 163s 6s/step - loss: 0.0631 -
accuracy: 0.9945 - val_loss: 0.2081 - val_accuracy: 0.9163
Epoch 18/20
29/29 [==============================] - 157s 5s/step - loss: 0.0574 -
```

```
accuracy: 0.9962 - val_loss: 0.2072 - val_accuracy: 0.9212
Epoch 19/20
29/29 [==============================] - 156s 5s/step - loss: 0.0534 -
accuracy: 0.9945 - val_loss: 0.2119 - val_accuracy: 0.9163
Epoch 20/20
29/29 [==============================] - 142s 5s/step - loss: 0.0485 -
accuracy: 0.9962 - val_loss: 0.2109 - val_accuracy: 0.9261
8/8 [==============================] - 18s 2s/step - loss: 0.1989 -
accuracy: 0.9204
Loss: 0.19885849952697754 Accuracy: 0.9203540086746216
              precision    recall  f1-score   support

       daisy       1.00      0.94      0.97        49
   dandelion       0.84      0.87      0.85        30
       roses       0.94      0.88      0.91        52
  sunflowers       0.93      0.93      0.93        41
      tulips       0.88      0.96      0.92        54

    accuracy                           0.92       226
   macro avg       0.92      0.92      0.92       226
weighted avg       0.92      0.92      0.92       226
```



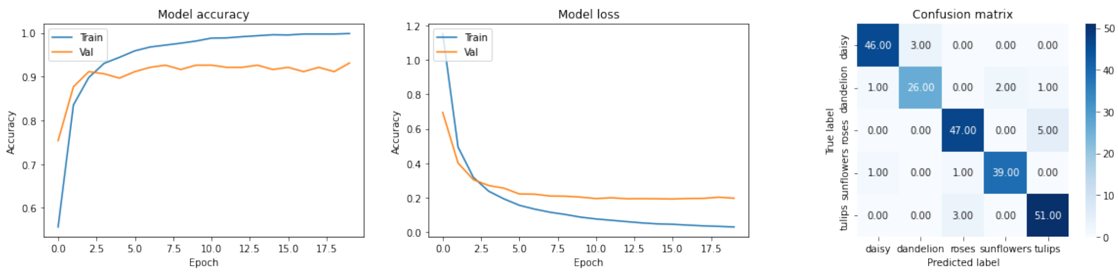




```python
model = train_save_evaluate(model_4, 'models\CNN_04.h5', X_train,
y_train, X_val, y_val, X_test, y_test, show_summary = False)
```

```
Epoch 1/20
29/29 [==============================] - 175s 6s/step - loss: 1.1525 -
accuracy: 0.5565 - val_loss: 0.6947 - val_accuracy: 0.7537
Epoch 2/20
29/29 [==============================] - 160s 6s/step - loss: 0.4936 -
accuracy: 0.8353 - val_loss: 0.4005 - val_accuracy: 0.8768
Epoch 3/20
29/29 [==============================] - 160s 6s/step - loss: 0.3175 -
accuracy: 0.8979 - val_loss: 0.3044 - val_accuracy: 0.9113
Epoch 4/20
29/29 [==============================] - 161s 6s/step - loss: 0.2371 -
accuracy: 0.9303 - val_loss: 0.2703 - val_accuracy: 0.9064
Epoch 5/20
29/29 [==============================] - 161s 6s/step - loss: 0.1925 -
accuracy: 0.9440 - val_loss: 0.2547 - val_accuracy: 0.8966
Epoch 6/20
```

```
29/29 [==============================] - 164s 6s/step - loss: 0.1555 -
accuracy: 0.9588 - val_loss: 0.2211 - val_accuracy: 0.9113
Epoch 7/20
29/29 [==============================] - 161s 6s/step - loss: 0.1332 -
accuracy: 0.9676 - val_loss: 0.2201 - val_accuracy: 0.9212
Epoch 8/20
29/29 [==============================] - 165s 6s/step - loss: 0.1153 -
accuracy: 0.9720 - val_loss: 0.2095 - val_accuracy: 0.9261
Epoch 9/20
29/29 [==============================] - 166s 6s/step - loss: 0.1022 -
accuracy: 0.9764 - val_loss: 0.2084 - val_accuracy: 0.9163
Epoch 10/20
29/29 [==============================] - 163s 6s/step - loss: 0.0867 -
accuracy: 0.9813 - val_loss: 0.2031 - val_accuracy: 0.9261
Epoch 11/20
29/29 [==============================] - 165s 6s/step - loss: 0.0763 -
accuracy: 0.9879 - val_loss: 0.1947 - val_accuracy: 0.9261
Epoch 12/20
29/29 [==============================] - 163s 6s/step - loss: 0.0687 -
accuracy: 0.9885 - val_loss: 0.1992 - val_accuracy: 0.9212
Epoch 13/20
29/29 [==============================] - 166s 6s/step - loss: 0.0609 -
accuracy: 0.9912 - val_loss: 0.1936 - val_accuracy: 0.9212
Epoch 14/20
29/29 [==============================] - 164s 6s/step - loss: 0.0534 -
accuracy: 0.9934 - val_loss: 0.1941 - val_accuracy: 0.9261
Epoch 15/20
29/29 [==============================] - 162s 6s/step - loss: 0.0479 -
accuracy: 0.9956 - val_loss: 0.1934 - val_accuracy: 0.9163
Epoch 16/20
29/29 [==============================] - 163s 6s/step - loss: 0.0455 -
accuracy: 0.9951 - val_loss: 0.1922 - val_accuracy: 0.9212
Epoch 17/20
29/29 [==============================] - 163s 6s/step - loss: 0.0406 -
accuracy: 0.9973 - val_loss: 0.1951 - val_accuracy: 0.9113
Epoch 18/20
29/29 [==============================] - 165s 6s/step - loss: 0.0364 -
accuracy: 0.9973 - val_loss: 0.1958 - val_accuracy: 0.9212
Epoch 19/20
29/29 [==============================] - 171s 6s/step - loss: 0.0333 -
accuracy: 0.9973 - val_loss: 0.2026 - val_accuracy: 0.9113
Epoch 20/20
29/29 [==============================] - 171s 6s/step - loss: 0.0299 -
accuracy: 0.9984 - val_loss: 0.1967 - val_accuracy: 0.9310
8/8 [==============================] - 21s 3s/step - loss: 0.1962 -
accuracy: 0.9248
Loss: 0.1961650401353836 Accuracy: 0.9247787594795227
              precision    recall  f1-score   support

       daisy       0.96      0.94      0.95        49
```

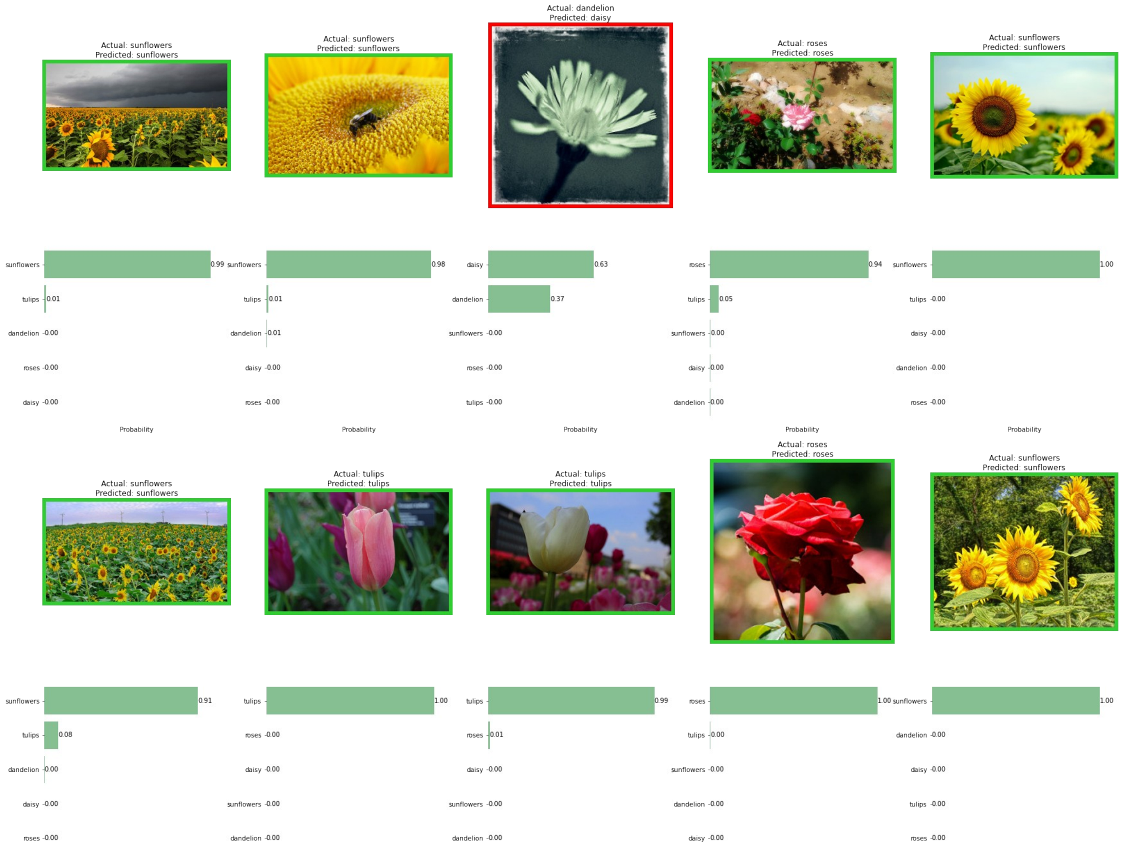
```
   dandelion       0.90      0.87      0.88        30
       roses       0.92      0.90      0.91        52
  sunflowers       0.95      0.95      0.95        41
      tulips       0.89      0.94      0.92        54

    accuracy                           0.92       226
   macro avg       0.92      0.92      0.92       226
weighted avg       0.93      0.92      0.92       226
```




## Best CNN model

The best models are based on DenseNet201 and have accuracy about 90%. Let's upload it and view the results.

```
path = 'models\CNN_04.h5'
model = tf.keras.models.load_model(path)

idxs = np.random.randint(0, len(X_init), size=10)
imgs = [X_init[i] for i in idxs]
y_test_imgs = [y_init[i] for i in idxs]
plot_inference(inference_cnn, model, imgs, y_test_imgs, with_prob =
True)
```




## ANN

After investigating different articles I have tried to do something similar but unfortunately the accuracy of models were poor, the best one has only 40% :(

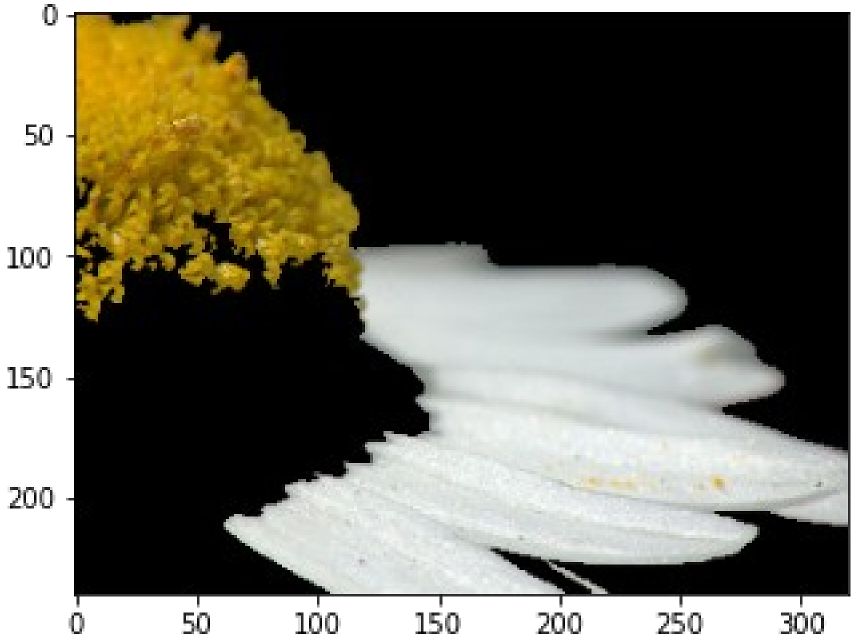
Ideas which I have been trying were:

- use segmentation (with thresholding and finding the biggest area as image), then calculate HOG, mean and std for color channels
- blur images a bit with morphology, then calculate HOG
- using Gabor features

```
# Image segmented -> color mean, std, hog -> 40%
# Morphology, resize -> hog -> 42%
def model_5():
    model = Sequential()
    #model.add(Flatten(input_shape=X_train.shape[1:]))
    model.add(Dense(32, activation='relu',
input_shape=X_train.shape[1:]))
    #model.add(Dense(32, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(32, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(CLASS_NUM,  activation='softmax'))
```

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

def largest_component_mask(bin_img):
    """Finds the largest component in a binary image and returns the
component as a mask."""

    contours = cv2.findContours(bin_img, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_NONE)[0]
    # should be [1] if OpenCV 3+

    max_area = 0
    max_contour_index = 0
    for i, contour in enumerate(contours):
        contour_area = cv2.moments(contour)['m00']
        if contour_area > max_area:
            max_area = contour_area
            max_contour_index = i

    labeled_img = np.zeros(bin_img.shape, dtype=np.uint8)
    cv2.drawContours(labeled_img, contours, max_contour_index,
color=255, thickness=-1)

    return labeled_img

def segment(image):
    img = image.copy()
    img = cv2.cvtColor(img,cv2.COLOR_RGB2GRAY)
    img =
cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_MEAN_C,cv2.THRESH_BI
NARY,255,2)
    mask = largest_component_mask(img)
    res = cv2.bitwise_and(image,image,mask = mask)
    return res

img = X_init[200]
res = segment(img)
plt.imshow(res, cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x1ad6e9711f0>
```




```
DIM = (128, 128)
def preprocess(X_init):
    X = []
    for i in range(len(X_init)):
        img = X_init[i].copy()
        kernel = np.ones((3,3),np.uint8)
        img = cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel,
iterations=2)
        img = cv2.resize(img, DIM, interpolation = cv2.INTER_AREA)
        #img = cv2.cvtColor(img,cv2.COLOR_RGB2GRAY)
        img = segment(img)
        X.append(img)
    return X

X = preprocess(X_init)
X = np.array(X)
plot_sample(X, y)

def preprocess(X_init):
    X = []
    hog = cv2.HOGDescriptor()
    for i in range(len(X_init)):
        img = X_init[i].copy()
        h = hog.compute(img)
        color = img.mean(), img.std()
        res = np.concatenate((h, color), axis=None)
        X.append(res)
    return X
X = preprocess(X)
X = np.array(X)
```


```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.1, random_state=0) # split data
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train,
test_size=0.1, random_state=0) # split data
print("Train size:", len(X_train))
print("Val size:", len(X_val))
print("Test size:", len(X_test))
```

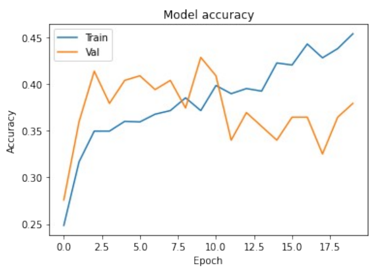
```
Train size: 1822
Val size: 203
Test size: 226
```

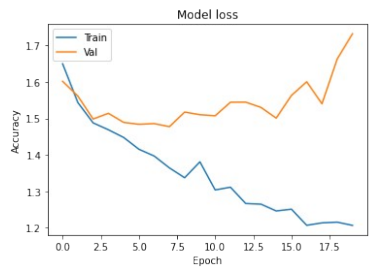
```
model = train_save_evaluate(model_5, 'models\ANN_01.h5', X_train,
y_train, X_val, y_val, X_test, y_test)
```

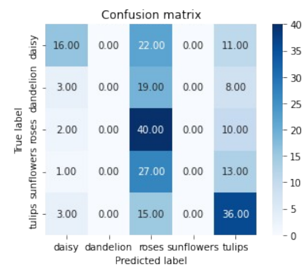
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 32)                1088736

 dropout (Dropout)           (None, 32)                0

 dense_1 (Dense)             (None, 32)                1056

 dropout_1 (Dropout)         (None, 32)                0

 dense_2 (Dense)             (None, 5)                 165

=================================================================
Total params: 1,089,957
Trainable params: 1,089,957
Non-trainable params: 0
_________________________________________________________________
Epoch 1/20
29/29 [==============================] - 1s 23ms/step - loss: 1.6496 -
accuracy: 0.2486 - val_loss: 1.6014 - val_accuracy: 0.2759
Epoch 2/20
29/29 [==============================] - 0s 15ms/step - loss: 1.5432 -
accuracy: 0.3167 - val_loss: 1.5610 - val_accuracy: 0.3596
Epoch 3/20
29/29 [==============================] - 0s 16ms/step - loss: 1.4876 -
accuracy: 0.3496 - val_loss: 1.4981 - val_accuracy: 0.4138
Epoch 4/20
```

```
29/29 [==============================] - 0s 16ms/step - loss: 1.4687 -
accuracy: 0.3496 - val_loss: 1.5138 - val_accuracy: 0.3793
Epoch 5/20
29/29 [==============================] - 0s 16ms/step - loss: 1.4477 -
accuracy: 0.3600 - val_loss: 1.4889 - val_accuracy: 0.4039
Epoch 6/20
29/29 [==============================] - 0s 15ms/step - loss: 1.4153 -
accuracy: 0.3595 - val_loss: 1.4836 - val_accuracy: 0.4089
Epoch 7/20
29/29 [==============================] - 0s 16ms/step - loss: 1.3968 -
accuracy: 0.3677 - val_loss: 1.4857 - val_accuracy: 0.3941
Epoch 8/20
29/29 [==============================] - 0s 16ms/step - loss: 1.3642 -
accuracy: 0.3716 - val_loss: 1.4772 - val_accuracy: 0.4039
Epoch 9/20
29/29 [==============================] - 0s 15ms/step - loss: 1.3372 -
accuracy: 0.3853 - val_loss: 1.5172 - val_accuracy: 0.3744
Epoch 10/20
29/29 [==============================] - 1s 18ms/step - loss: 1.3808 -
accuracy: 0.3716 - val_loss: 1.5100 - val_accuracy: 0.4286
Epoch 11/20
29/29 [==============================] - 0s 16ms/step - loss: 1.3037 -
accuracy: 0.3985 - val_loss: 1.5070 - val_accuracy: 0.4089
Epoch 12/20
29/29 [==============================] - 0s 17ms/step - loss: 1.3113 -
accuracy: 0.3897 - val_loss: 1.5442 - val_accuracy: 0.3399
Epoch 13/20
29/29 [==============================] - 0s 16ms/step - loss: 1.2666 -
accuracy: 0.3952 - val_loss: 1.5444 - val_accuracy: 0.3695
Epoch 14/20
29/29 [==============================] - 1s 17ms/step - loss: 1.2648 -
accuracy: 0.3924 - val_loss: 1.5300 - val_accuracy: 0.3547
Epoch 15/20
29/29 [==============================] - 0s 17ms/step - loss: 1.2461 -
accuracy: 0.4226 - val_loss: 1.5006 - val_accuracy: 0.3399
Epoch 16/20
29/29 [==============================] - 0s 17ms/step - loss: 1.2509 -
accuracy: 0.4204 - val_loss: 1.5628 - val_accuracy: 0.3645
Epoch 17/20
29/29 [==============================] - 0s 17ms/step - loss: 1.2067 -
accuracy: 0.4429 - val_loss: 1.6003 - val_accuracy: 0.3645
Epoch 18/20
29/29 [==============================] - 0s 16ms/step - loss: 1.2135 -
accuracy: 0.4281 - val_loss: 1.5400 - val_accuracy: 0.3251
Epoch 19/20
29/29 [==============================] - 0s 16ms/step - loss: 1.2154 -
accuracy: 0.4380 - val_loss: 1.6628 - val_accuracy: 0.3645
Epoch 20/20
29/29 [==============================] - 0s 16ms/step - loss: 1.2066 -
accuracy: 0.4539 - val_loss: 1.7320 - val_accuracy: 0.3793
```

```
8/8 [==============================] - 0s 5ms/step - loss: 1.7433 -
accuracy: 0.4071
Loss: 1.7433063983917236 Accuracy: 0.40707963705062866
              precision    recall  f1-score   support

       daisy       0.64      0.33      0.43        49
   dandelion       0.00      0.00      0.00        30
       roses       0.33      0.77      0.46        52
  sunflowers       0.00      0.00      0.00        41
      tulips       0.46      0.67      0.55        54

    accuracy                           0.41       226
   macro avg       0.29      0.35      0.29       226
weighted avg       0.32      0.41      0.33       226
```




## Other snippets

Maybe I will try later to deal with ANN

```
def preprocess(X_init):
    X = []
    for i in range(len(X_init)):
        img = X_init[i].copy()
        img = cv2.resize(img, DIM, interpolation = cv2.INTER_AREA)
        img = cv2.blur(img,(5,5))
        img = cv2.cvtColor(img,cv2.COLOR_RGB2HSV)
        X.append(img)
    return X

def preprocess(X_init):
    X = []
    for i in range(len(X_init)):
        img = X_init[i].copy()
        #img = cv2.blur(img,(5,5))
        kernel = np.ones((3,3),np.uint8)
        img = cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel,
iterations=2)
        img = cv2.resize(img, DIM, interpolation = cv2.INTER_AREA)
        img = cv2.cvtColor(img,cv2.COLOR_RGB2GRAY)
        #img = segment(img)
```

```python
        X.append(img)
    return X

dim = (128, 128)
def preprocess(X_init):
    X = []
    hog = cv2.HOGDescriptor()
    for i in range(len(X_init)):
        img = X_init[i].copy()
        img = cv2.resize(img, dim, interpolation = cv2.INTER_AREA)
        #img = cv2.blur(img,(5,5))
        h = hog.compute(img)
        X.append(h)
    return X
X = preprocess(X2)

def preprocess(X_init):
    X = []
    hog = cv2.HOGDescriptor()
    for i in range(len(X_init)):
        img = X_init[i].copy()
        h = hog.compute(img)
        #color = np.array(img).flatten()
        color = img.mean(), img.std()
        res = np.concatenate((h, color), axis=None)
        X.append(res)
    return X
X = preprocess(X_init)

import cv2
import numpy as np
import pylab as pl
import glob
import pandas as pd

# define gabor filter bank with different orientations and at
different scales
def build_filters():
    filters = []
    ksize = 9
    #define the range for theta and nu
    for theta in np.arange(0, np.pi, np.pi / 8):
        for nu in np.arange(0, 6*np.pi/4 , np.pi / 4):
            kern = cv2.getGaborKernel((ksize, ksize), 1.0, theta, nu,
0.5, 0, ktype=cv2.CV_32F)
            kern /= 1.5*kern.sum()
            filters.append(kern)
    return filters

#function to convolve the image with the filters
def process(img, filters):
```

```
    accum = np.zeros_like(img)
    for kern in filters:
        fimg = cv2.filter2D(img, cv2.CV_8UC3, kern)
        np.maximum(accum, fimg, accum)
    return accum

def preprocess_img(img, filters):
    feat = []
    #calculating the local energy for each convolved image
    for j in range(40):
        res = process(img, f[j])
        temp = 0
        for p in range(128):
            for q in range(128):
                temp = temp + res[p][q]*res[p][q]
        feat.append(temp)
    #calculating the mean amplitude for each convolved image
    for j in range(40):
        res = process(img, f[j])
        temp = 0
        for p in range(128):
            for q in range(128):
                temp = temp + abs(res[p][q])
        feat.append(temp)
     #feat matrix is the feature vector for the image
    return feat

filters = build_filters()
f = np.asarray(filters)

#img = cv2.imread(path)
#img = cv2.resize(img, dim, interpolation = cv2.INTER_AREA)
#feats = preprocess_img(img, f)

dim = (128, 128)
def preprocess(X_init):
    X = []
    filters = build_filters()
    f = np.asarray(filters)
    for i in range(len(X_init)):
        img = X_init[i].copy()
        img = cv2.resize(img, dim, interpolation = cv2.INTER_AREA)
        feats = preprocess_img(img, f)
        X.append(feats)
    return X
X = preprocess(X_init)
```