alexnet-implementation

March 12, 2024

[1]:
```
# Importing Keras libraries and packages
from keras.models import Sequential
from keras.layers import Convolution2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import BatchNormalization
```

```
2024-03-12 15:30:43.124783: E
external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:9261] Unable to register
cuDNN factory: Attempting to register factory for plugin cuDNN when one has
already been registered
2024-03-12 15:30:43.124896: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:607] Unable to register
cuFFT factory: Attempting to register factory for plugin cuFFT when one has
already been registered
2024-03-12 15:30:43.258044: E
external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1515] Unable to
register cuBLAS factory: Attempting to register factory for plugin cuBLAS when
one has already been registered
```

[2]:
```
# Initializing the CNN
classifier = Sequential()

# Convolution Step 1
classifier.add(Convolution2D(96, 11, strides = (4, 4), padding = 'valid',␣
↪input_shape=(224, 224, 3), activation = 'relu'))

# Max Pooling Step 1
classifier.add(MaxPooling2D(pool_size = (2, 2), strides = (2, 2), padding =␣
↪'valid'))
classifier.add(BatchNormalization())

# Convolution Step 2
classifier.add(Convolution2D(256, 11, strides = (1, 1), padding='valid',␣
↪activation = 'relu'))
```

```
# Max Pooling Step 2
classifier.add(MaxPooling2D(pool_size = (2, 2), strides = (2, 2),␣
 ↪padding='valid'))
classifier.add(BatchNormalization())

# Convolution Step 3
classifier.add(Convolution2D(384, 3, strides = (1, 1), padding='valid',␣
 ↪activation = 'relu'))
classifier.add(BatchNormalization())

# Convolution Step 4
classifier.add(Convolution2D(384, 3, strides = (1, 1), padding='valid',␣
 ↪activation = 'relu'))
classifier.add(BatchNormalization())

# Convolution Step 5
classifier.add(Convolution2D(256, 3, strides=(1,1), padding='valid', activation␣
 ↪= 'relu'))

# Max Pooling Step 3
classifier.add(MaxPooling2D(pool_size = (2, 2), strides = (2, 2), padding =␣
 ↪'valid'))
classifier.add(BatchNormalization())

# Flattening Step
classifier.add(Flatten())

# Full Connection Step
classifier.add(Dense(units = 4096, activation = 'relu'))
classifier.add(Dropout(0.4))
classifier.add(BatchNormalization())
classifier.add(Dense(units = 4096, activation = 'relu'))
classifier.add(Dropout(0.4))
classifier.add(BatchNormalization())
classifier.add(Dense(units = 1000, activation = 'relu'))
classifier.add(Dropout(0.2))
classifier.add(BatchNormalization())
classifier.add(Dense(units = 38, activation = 'softmax'))
classifier.summary()
```

```
/opt/conda/lib/python3.10/site-
packages/keras/src/layers/convolutional/base_conv.py:99: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 54, 54, 96)	34,944
max_pooling2d (MaxPooling2D)	(None, 27, 27, 96)	0
batch_normalization (BatchNormalization)	(None, 27, 27, 96)	384
conv2d_1 (Conv2D)	(None, 17, 17, 256)	2,973,952
max_pooling2d_1 (MaxPooling2D)	(None, 8, 8, 256)	0
batch_normalization_1 (BatchNormalization)	(None, 8, 8, 256)	1,024
conv2d_2 (Conv2D)	(None, 6, 6, 384)	885,120
batch_normalization_2 (BatchNormalization)	(None, 6, 6, 384)	1,536
conv2d_3 (Conv2D)	(None, 4, 4, 384)	1,327,488
batch_normalization_3 (BatchNormalization)	(None, 4, 4, 384)	1,536
conv2d_4 (Conv2D)	(None, 2, 2, 256)	884,992
max_pooling2d_2 (MaxPooling2D)	(None, 1, 1, 256)	0
batch_normalization_4 (BatchNormalization)	(None, 1, 1, 256)	1,024
flatten (Flatten)	(None, 256)	0
dense (Dense)	(None, 4096)	1,052,672
dropout (Dropout)	(None, 4096)	0
batch_normalization_5 (BatchNormalization)	(None, 4096)	16,384
dense_1 (Dense)	(None, 4096)	16,781,312

```
dropout_1 (Dropout)                  (None, 4096)                        0

batch_normalization_6                (None, 4096)                   16,384
(BatchNormalization)

dense_2 (Dense)                      (None, 1000)                4,097,000

dropout_2 (Dropout)                  (None, 1000)                        0

batch_normalization_7                (None, 1000)                   4,000
(BatchNormalization)

dense_3 (Dense)                      (None, 38)                     38,038
```

Total params: 28,117,790 (107.26 MB)

Trainable params: 28,096,654 (107.18 MB)

Non-trainable params: 21,136 (82.56 KB)

[3]:
```
classifier.load_weights('/kaggle/input/bestdata/best_weights_9.hdf5')
```

[4]:
```
# visualize layer names and layer indices to see how many layers
from keras import layers
for i, layer in enumerate(classifier.layers):
   print(i, layer.name)
```

```
0 conv2d
1 max_pooling2d
2 batch_normalization
3 conv2d_1
4 max_pooling2d_1
5 batch_normalization_1
6 conv2d_2
7 batch_normalization_2
8 conv2d_3
9 batch_normalization_3
10 conv2d_4
11 max_pooling2d_2
12 batch_normalization_4
13 flatten
14 dense
15 dropout
16 batch_normalization_5
```

```
17 dense_1
18 dropout_1
19 batch_normalization_6
20 dense_2
21 dropout_2
22 batch_normalization_7
23 dense_3
```

[5]:
```
# we chose to train the top 2 conv blocks, i.e. we will freeze
# the first 8 layers and unfreeze the rest:
print("Freezed layers:")
for i, layer in enumerate(classifier.layers[:20]):
    print(i, layer.name)
    layer.trainable = False
```

```
Freezed layers:
0 conv2d
1 max_pooling2d
2 batch_normalization
3 conv2d_1
4 max_pooling2d_1
5 batch_normalization_1
6 conv2d_2
7 batch_normalization_2
8 conv2d_3
9 batch_normalization_3
10 conv2d_4
11 max_pooling2d_2
12 batch_normalization_4
13 flatten
14 dense
15 dropout
16 batch_normalization_5
17 dense_1
18 dropout_1
19 batch_normalization_6
```

[6]:
```
#trainable parameters decrease after freezing some bottom layers
classifier.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 54, 54, 96)	34,944

max_pooling2d (MaxPooling2D)	(None, 27, 27, 96)	0
batch_normalization (BatchNormalization)	(None, 27, 27, 96)	384
conv2d_1 (Conv2D)	(None, 17, 17, 256)	2,973,952
max_pooling2d_1 (MaxPooling2D)	(None, 8, 8, 256)	0
batch_normalization_1 (BatchNormalization)	(None, 8, 8, 256)	1,024
conv2d_2 (Conv2D)	(None, 6, 6, 384)	885,120
batch_normalization_2 (BatchNormalization)	(None, 6, 6, 384)	1,536
conv2d_3 (Conv2D)	(None, 4, 4, 384)	1,327,488
batch_normalization_3 (BatchNormalization)	(None, 4, 4, 384)	1,536
conv2d_4 (Conv2D)	(None, 2, 2, 256)	884,992
max_pooling2d_2 (MaxPooling2D)	(None, 1, 1, 256)	0
batch_normalization_4 (BatchNormalization)	(None, 1, 1, 256)	1,024
flatten (Flatten)	(None, 256)	0
dense (Dense)	(None, 4096)	1,052,672
dropout (Dropout)	(None, 4096)	0
batch_normalization_5 (BatchNormalization)	(None, 4096)	16,384
dense_1 (Dense)	(None, 4096)	16,781,312
dropout_1 (Dropout)	(None, 4096)	0
batch_normalization_6 (BatchNormalization)	(None, 4096)	16,384
dense_2 (Dense)	(None, 1000)	4,097,000
dropout_2 (Dropout)	(None, 1000)	0

```
batch_normalization_7          (None, 1000)              4,000
(BatchNormalization)

dense_3 (Dense)                (None, 38)                38,038
```

Total params: 28,117,790 (107.26 MB)

Trainable params: 4,137,038 (15.78 MB)

Non-trainable params: 23,980,752 (91.48 MB)

[7]:
```
from keras import optimizers

# Initialize the SGD optimizer with supported arguments (lr for learning rate,␣
 ↪momentum)
optimizer = optimizers.SGD(learning_rate=0.001, momentum=0.9)

# Compile the model with the initialized optimizer
classifier.compile(optimizer=optimizer,
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

[8]:
```
# image preprocessing
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale=1./255,
                                   shear_range=0.2,
                                   zoom_range=0.2,
                                   width_shift_range=0.2,
                                   height_shift_range=0.2,
                                   fill_mode='nearest')

valid_datagen = ImageDataGenerator(rescale=1./255)

batch_size = 128
base_dir = "../input/new-plant-diseases-dataset/new plant diseases␣
 ↪dataset(augmented)/New Plant Diseases Dataset(Augmented)"

training_set = train_datagen.flow_from_directory(base_dir+'/train',
                                                 target_size=(224, 224),
                                                 batch_size=batch_size,
                                                 class_mode='categorical')
```

```
valid_set = valid_datagen.flow_from_directory(base_dir+'/valid',
                                              target_size=(224, 224),
                                              batch_size=batch_size,
                                              class_mode='categorical')
```

```
Found 70295 images belonging to 38 classes.
Found 17572 images belonging to 38 classes.
```

[9]:
```
class_dict = training_set.class_indices
print(class_dict)
```

```
{'Apple___Apple_scab': 0, 'Apple___Black_rot': 1, 'Apple___Cedar_apple_rust': 2,
'Apple___healthy': 3, 'Blueberry___healthy': 4,
'Cherry_(including_sour)___Powdery_mildew': 5,
'Cherry_(including_sour)___healthy': 6, 'Corn_(maize)___Cercospora_leaf_spot
Gray_leaf_spot': 7, 'Corn_(maize)___Common_rust_': 8,
'Corn_(maize)___Northern_Leaf_Blight': 9, 'Corn_(maize)___healthy': 10,
'Grape___Black_rot': 11, 'Grape___Esca_(Black_Measles)': 12,
'Grape___Leaf_blight_(Isariopsis_Leaf_Spot)': 13, 'Grape___healthy': 14,
'Orange___Haunglongbing_(Citrus_greening)': 15, 'Peach___Bacterial_spot': 16,
'Peach___healthy': 17, 'Pepper,_bell___Bacterial_spot': 18,
'Pepper,_bell___healthy': 19, 'Potato___Early_blight': 20,
'Potato___Late_blight': 21, 'Potato___healthy': 22, 'Raspberry___healthy': 23,
'Soybean___healthy': 24, 'Squash___Powdery_mildew': 25,
'Strawberry___Leaf_scorch': 26, 'Strawberry___healthy': 27,
'Tomato___Bacterial_spot': 28, 'Tomato___Early_blight': 29,
'Tomato___Late_blight': 30, 'Tomato___Leaf_Mold': 31,
'Tomato___Septoria_leaf_spot': 32, 'Tomato___Spider_mites Two-
spotted_spider_mite': 33, 'Tomato___Target_Spot': 34,
'Tomato___Tomato_Yellow_Leaf_Curl_Virus': 35, 'Tomato___Tomato_mosaic_virus':
36, 'Tomato___healthy': 37}
```

[10]:
```
li = list(class_dict.keys())
print(li)
```

```
['Apple___Apple_scab', 'Apple___Black_rot', 'Apple___Cedar_apple_rust',
'Apple___healthy', 'Blueberry___healthy',
'Cherry_(including_sour)___Powdery_mildew', 'Cherry_(including_sour)___healthy',
'Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot',
'Corn_(maize)___Common_rust_', 'Corn_(maize)___Northern_Leaf_Blight',
'Corn_(maize)___healthy', 'Grape___Black_rot', 'Grape___Esca_(Black_Measles)',
'Grape___Leaf_blight_(Isariopsis_Leaf_Spot)', 'Grape___healthy',
'Orange___Haunglongbing_(Citrus_greening)', 'Peach___Bacterial_spot',
'Peach___healthy', 'Pepper,_bell___Bacterial_spot', 'Pepper,_bell___healthy',
'Potato___Early_blight', 'Potato___Late_blight', 'Potato___healthy',
'Raspberry___healthy', 'Soybean___healthy', 'Squash___Powdery_mildew',
'Strawberry___Leaf_scorch', 'Strawberry___healthy', 'Tomato___Bacterial_spot',
'Tomato___Early_blight', 'Tomato___Late_blight', 'Tomato___Leaf_Mold',
```

```
'Tomato___Septoria_leaf_spot', 'Tomato___Spider_mites Two-spotted_spider_mite',
'Tomato___Target_Spot', 'Tomato___Tomato_Yellow_Leaf_Curl_Virus',
'Tomato___Tomato_mosaic_virus', 'Tomato___healthy']
```

[11]:
```
train_num = training_set.samples
valid_num = valid_set.samples
```

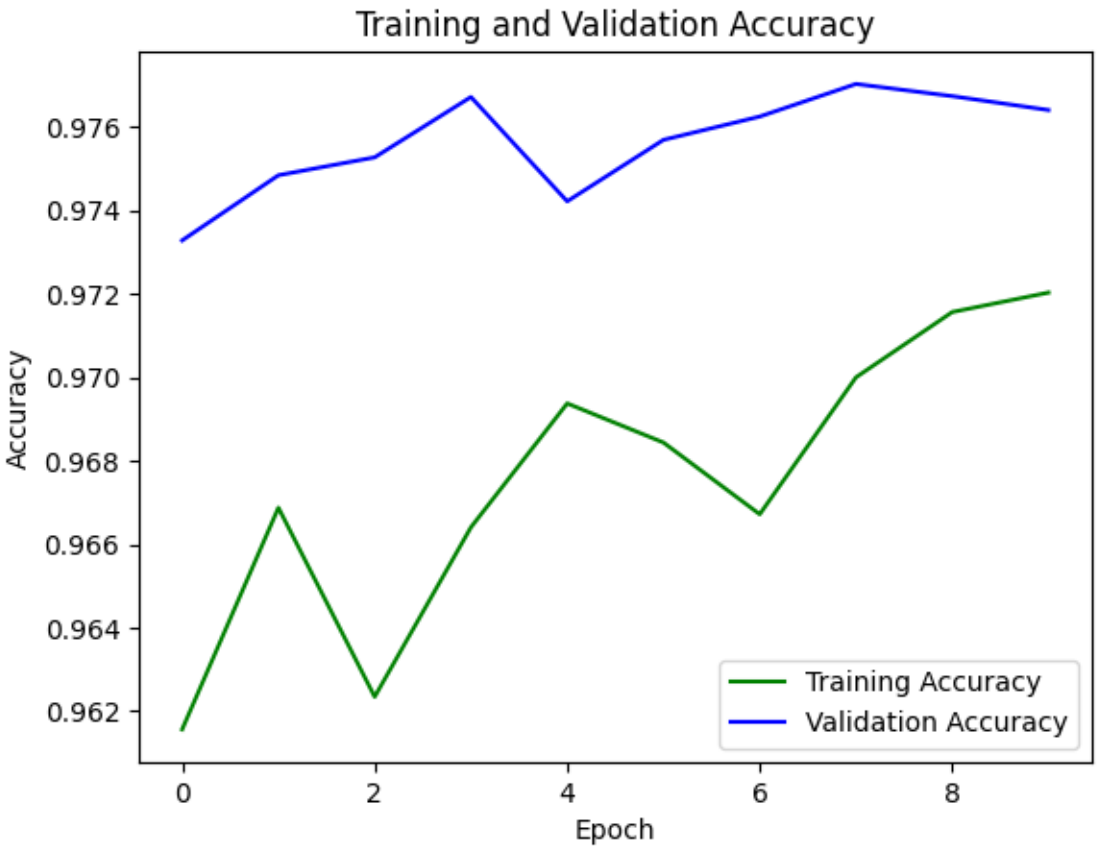
[12]:
```
from keras.callbacks import ModelCheckpoint

weightpath = "/kaggle/input/bestdata/best_weights_9.weights.h5"
checkpoint = ModelCheckpoint(weightpath,
                             monitor='val_acc',
                             verbose=1,
                             save_best_only=True,
                             save_weights_only=True,
                             mode='max')
callbacks_list = [checkpoint]
```

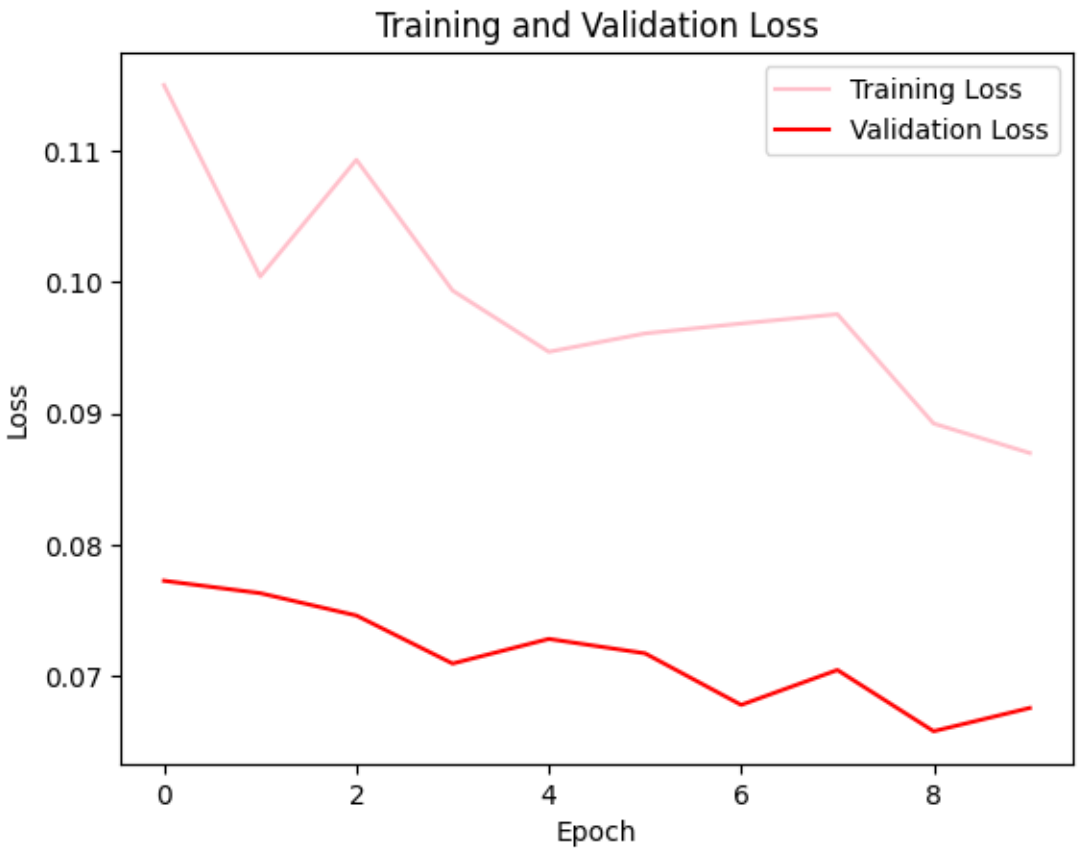
[19]:
```
history = classifier.fit(training_set,
                         steps_per_epoch=min(train_num // batch_size, 50),
                         validation_data=valid_set,
                         epochs=10,
                         validation_steps=min(valid_num // batch_size, 50),
                         callbacks=callbacks_list)
```

```
Epoch 1/10
50/50             185s 3s/step -
accuracy: 0.9601 - loss: 0.1232 - val_accuracy: 0.9733 - val_loss: 0.0773
Epoch 2/10
50/50             148s 3s/step -
accuracy: 0.9681 - loss: 0.0982 - val_accuracy: 0.9748 - val_loss: 0.0763
Epoch 3/10
50/50             123s 3s/step -
accuracy: 0.9648 - loss: 0.1029 - val_accuracy: 0.9753 - val_loss: 0.0746
Epoch 4/10
 3/50             1s 37ms/step -
accuracy: 0.9740 - loss: 0.0790

W0000 00:00:1710258429.278319      85 graph_launch.cc:671] Fallback to op-by-op
mode because memset node breaks graph update
/opt/conda/lib/python3.10/contextlib.py:153: UserWarning: Your input ran out of
data; interrupting training. Make sure that your dataset or generator can
generate at least `steps_per_epoch * epochs` batches. You may need to use the
`.repeat()` function when building your dataset.
  self.gen.throw(typ, value, traceback)

50/50             117s 2s/step -
accuracy: 0.9682 - loss: 0.0963 - val_accuracy: 0.9767 - val_loss: 0.0710
Epoch 5/10
```

```
50/50             98s 2s/step -
accuracy: 0.9674 - loss: 0.0941 - val_accuracy: 0.9742 - val_loss: 0.0728
Epoch 6/10
50/50             93s 2s/step -
accuracy: 0.9693 - loss: 0.0963 - val_accuracy: 0.9757 - val_loss: 0.0717
Epoch 7/10
50/50             103s 2s/step -
accuracy: 0.9695 - loss: 0.0926 - val_accuracy: 0.9762 - val_loss: 0.0678
Epoch 8/10
50/50             94s 2s/step -
accuracy: 0.9719 - loss: 0.0921 - val_accuracy: 0.9770 - val_loss: 0.0705
Epoch 9/10
50/50             86s 2s/step -
accuracy: 0.9744 - loss: 0.0870 - val_accuracy: 0.9767 - val_loss: 0.0658
Epoch 10/10
50/50             99s 2s/step -
accuracy: 0.9676 - loss: 0.0902 - val_accuracy: 0.9764 - val_loss: 0.0676
```

[20]:
```python
import matplotlib.pyplot as plt

# Plotting training and validation accuracy
plt.plot(history.history['accuracy'], label='Training Accuracy', color='green')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy',␣
 ↪color='blue')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Plotting training and validation loss
plt.plot(history.history['loss'], label='Training Loss', color='pink')
plt.plot(history.history['val_loss'], label='Validation Loss', color='red')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```


Training and Validation Accuracy
Accuracy
Epoch
0.976
0.974
0.972
0.970
0.968
0.966
0.964
0.962
0
2
4
6
8
Training Accuracy
Validation Accuracy



[21]:
```
# predicting an image
from keras.preprocessing import image
import numpy as np
image_path = "/kaggle/input/new-plant-diseases-dataset/test/test/
 ↪CornCommonRust3.JPG"
new_img = image.load_img(image_path, target_size=(224, 224))
img = image.img_to_array(new_img)
img = np.expand_dims(img, axis=0)
img = img/255

print("Following is our prediction:")
prediction = classifier.predict(img)
d = prediction.flatten()
j = d.max()
for index,item in enumerate(d):
    if item == j:
        class_name = li[index]


#ploting image with predicted class name
```

```
plt.figure(figsize = (4,4))
plt.imshow(new_img)
plt.axis('off')
plt.title(class_name)
plt.show()
```

```
Following is our prediction:
1/1             1s 1s/step
```