Image classification Using CNN and Transfer learning using VGG16

Muhammad Tamjid Rahman

Contents

Loading R packages

```
import tensorflow as tf
from tensorflow import keras
from keras import layers
from keras import models
from tensorflow.keras.optimizers import Adam
from keras.layers import BatchNormalization
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D
import itertools
from tensorflow.keras.utils import to_categorical
import numpy as np
import matplotlib.pyplot as plt
```

1 Convolutional neural networks using Keras

1)

Loading MNIST dataset

```
from keras.datasets import mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

Implementing a Convolutional Neural Network

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(32, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64,activation='relu'))
model.add(layers.Dense(10,activation='softmax'))
```

```
model.summary()
```

```
## Model: "sequential"
## _________________________________________________________________
##  Layer (type)                 Output Shape              Param #
## =================================================================
##  conv2d (Conv2D)              (None, 26, 26, 32)        320
##
##  max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
##  )
##
##  conv2d_1 (Conv2D)            (None, 11, 11, 32)        9248
##
##  flatten (Flatten)            (None, 3872)              0
##
##  dense (Dense)                (None, 64)                247872
##
##  dense_1 (Dense)              (None, 10)                650
##
## =================================================================
## Total params: 258,090
## Trainable params: 258,090
## Non-trainable params: 0
## _________________________________________________________________
```

2)

Here the kernel weight is 3x3 matrix in each channel and there are 32 channels. So, 3x3x32=288 weight parameters. For each channel there is a bias. So, 32 biases. So, total parameters=288+32= 320

3)

```
train_images = train_images.reshape((60000, 28, 28, 1))
train_images = train_images.astype('float32') / 255
test_images = test_images.reshape((10000, 28, 28, 1))
```

```
test_images = test_images.astype('float32') / 255
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

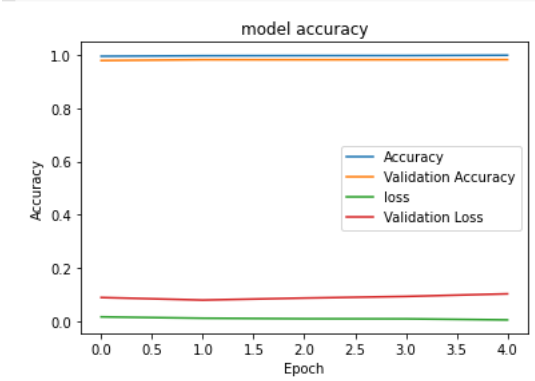
Separating validation dataset from train set and suffling

```
indices_permutation = np.random.permutation(len(train_images))
shuffled_images = train_images[indices_permutation]
shuffled_labels = train_labels[indices_permutation]
num_validation_samples = int(0.3 * len(train_images))
val_inputs = shuffled_images[-num_validation_samples:]
val_targets = shuffled_labels[-num_validation_samples:]
train_images = shuffled_images[:num_validation_samples]
train_labels = shuffled_labels[:num_validation_samples]
```

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

```
model.fit(train_images,train_labels,epochs=5,batch_size=16,validation_data=(val_inputs,val_targets))
```

```
plt.plot(history.history["accuracy"])
plt.plot(history.history['val_accuracy'])
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title("model accuracy")
plt.ylabel("Accuracy")
plt.xlabel("Epoch")
plt.legend(["Accuracy","Validation Accuracy","loss","Validation Loss"])
plt.show()
```




```
test_loss, test_acc = model.evaluate(test_images, test_labels)
```

```
313/313 [==============================] - 2s 5ms/step - loss: 0.0821 - accuracy: 0.9834
0.9833999872207642
```

```
test_acc
```

```
0.9833999872207642
```

```
test_loss
```

```
0.08208703249692917
```

4)

Loading CIFAR-10 dataset

```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()
x_train.shape == (50000, 32, 32, 3)
```

```
## True
```

```
x_test.shape == (10000, 32, 32, 3)
```

```
## True
```

```
y_train.shape == (50000, 1)
```

```
## True
```

```
assert y_test.shape == (10000, 1)
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

```
indices_permutation = np.random.permutation(len(x_train))
shuffled_x = x_train[indices_permutation]
shuffled_y = y_train[indices_permutation]
num_validation_samples = int(0.3 * len(x_train))
val_x = shuffled_x[-num_validation_samples:]
val_y = shuffled_y[-num_validation_samples:]
x_train = shuffled_x[:num_validation_samples]
y_train = shuffled_y[:num_validation_samples]
```

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64,activation='relu'))
model.add(layers.Dense(10,activation='softmax'))
```

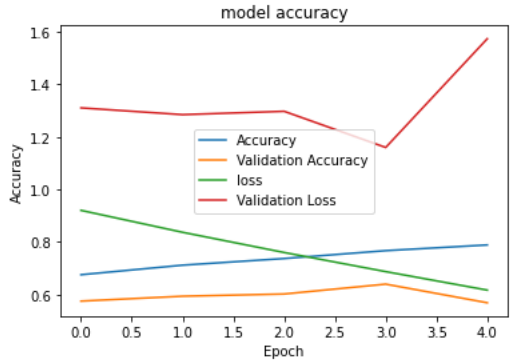
```
model.summary()
```

```
## Model: "sequential_1"
## _________________________________________________________________
##  Layer (type)                 Output Shape              Param #
## =================================================================
##  conv2d_2 (Conv2D)            (None, 30, 30, 32)        896
##
##  max_pooling2d_1 (MaxPooling  (None, 15, 15, 32)       0
##  2D)
##
##  conv2d_3 (Conv2D)            (None, 13, 13, 64)        18496
##
##  max_pooling2d_2 (MaxPooling  (None, 6, 6, 64)         0
##  2D)
```

```
##
##  conv2d_4 (Conv2D)           (None, 4, 4, 64)          36928
##
##  flatten_1 (Flatten)         (None, 1024)              0
##
##  dense_2 (Dense)             (None, 64)                65600
##
##  dense_3 (Dense)             (None, 10)                650
##
## =================================================================
## Total params: 122,570
## Trainable params: 122,570
## Non-trainable params: 0
## _________________________________________________________________
```

```
model.compile(optimizer='rmsprop',loss='categorical_crossentropy',metrics=['accuracy'])
history=model.fit(x_train, y_train, epochs=5, batch_size=16,validation_data=(val_x, val_y))
```

```
plt.plot(history.history["accuracy"])
plt.plot(history.history['val_accuracy'])
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title("model accuracy")
plt.ylabel("Accuracy")
plt.xlabel("Epoch")
plt.legend(["Accuracy","Validation Accuracy","loss","Validation Loss"])
plt.show()
```




```
test_loss, test_acc = model.evaluate(x_test, y_test)
```

```
313/313 [==============================] - 2s 5ms/step - loss: 1.5909 - accuracy: 0.5646
```

```
test_acc
```

```
0.5645999908447266
```

```
test_loss
```

```
1.5909385681152344
```

5)

Here the kernel weight is 3x3x3 matrix in each channel and there are 32 channels. So, 3x3x3x32=864 weight parameters. For each channel there is a bias. So, 32 biases. So, total parameters=864+32= 896

6) Testing three different networks

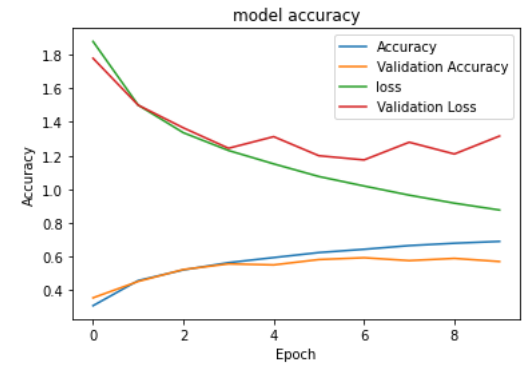
First model

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(32, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(32, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(32,activation='relu'))
model.add(layers.Dense(10,activation='softmax'))

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

history=model.fit(x_train, y_train, epochs=10, batch_size=16,validation_data=(val_x, val_y))
```

```
plt.plot(history.history["accuracy"])
plt.plot(history.history['val_accuracy'])
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title("model accuracy")
plt.ylabel("Accuracy")
plt.xlabel("Epoch")
plt.legend(["Accuracy","Validation Accuracy","loss","Validation Loss"])
plt.show()
```



We decrease the nods in every convolution layers without increasing any layer to build a simple and efficient model. We increase the epochs to 10 but after 3 epochs the models turns out to be over fitted. But We got a slightly better accuracy on the test data.

```
test_loss, test_acc = model.evaluate(x_test, y_test)
```

```
313/313 [==============================] - 1s 5ms/step - loss: 1.3162 - accuracy: 0.5684
```

```
test_acc
```

```
0.5684000253677368
```

```
test_loss
```

```
1.3161967992782593
```

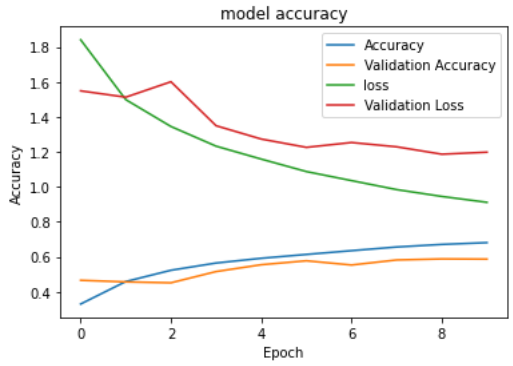
Second Model

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(32, (3, 3), activation='relu'))
model.add(Dropout(0.25))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(32, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(32,activation='relu'))
model.add(layers.Dense(10,activation='softmax'))

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

history=model.fit(x_train, y_train, epochs=10, batch_size=16,validation_data=(val_x, val_y))
```

```
plt.plot(history.history["accuracy"])
plt.plot(history.history['val_accuracy'])
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title("model accuracy")
plt.ylabel("Accuracy")
plt.xlabel("Epoch")
plt.legend(["Accuracy","Validation Accuracy","loss","Validation Loss"])
plt.show()
```



This time we add a Dropout layer in the second convolution layer without changing anything. After 5 epochs the model is over fitted but again out accuracy was improved a little.

```
test_loss, test_acc = model.evaluate(x_test, y_test)
```

```
313/313 [==============================] - 1s 5ms/step - loss: 1.2088 - accuracy: 0.5793
```

```
test_acc
```

```
0.5792999863624573
```

```
test_loss
```

```
1.208806037902832
```

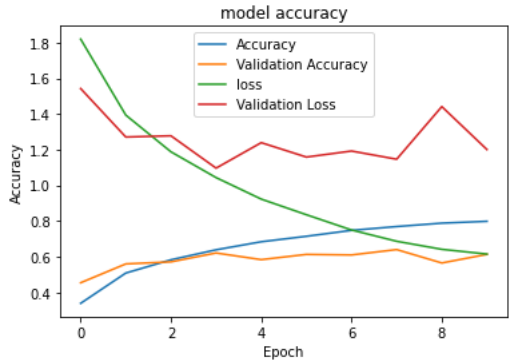
Third model

```
model = models.Sequential()
model.add(layers.Conv2D(128, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(Dropout(0.25))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64,activation='relu'))
model.add(layers.Dense(10,activation='softmax'))

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

history=model.fit(x_train, y_train, epochs=10, batch_size=16,validation_data=(val_x, val_y))
```

```
plt.plot(history.history["accuracy"])
plt.plot(history.history['val_accuracy'])
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title("model accuracy")
plt.ylabel("Accuracy")
plt.xlabel("Epoch")
plt.legend(["Accuracy","Validation Accuracy","loss","Validation Loss"])
plt.show()
```



We increased the nods in every hidden layers. The model was being over fitted after 3 epochs but the accuracy

for the test data was improved again.

```
test_loss, test_acc = model.evaluate(x_test, y_test)
```

```
313/313 [==============================] - 2s 7ms/step - loss: 1.2320 - accuracy: 0.6108
```

```
test_acc
```

```
0.61080002784729
```

```
test_loss
```

```
1.2319855690002441
```

6

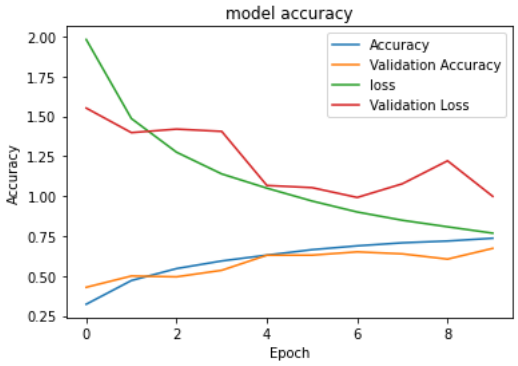
```
model = models.Sequential()
model.add(layers.Conv2D(32, (5,5), padding='same', activation='relu', input_shape=(32,32,3)))
model.add(layers.BatchNormalization())
model.add(layers.Conv2D(32, (3,3), padding='same', activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.AveragePooling2D(2,2))
model.add(layers.Dropout(0.3))
model.add(layers.Conv2D(64, (5,5),strides=1, padding='valid', activation='tanh'))
model.add(layers.BatchNormalization())
model.add(layers.Conv2D(64, (3,3), padding='same', activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3,3), padding='same', activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.AveragePooling2D(2,2))
model.add(layers.Dropout(0.3))

model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.Dropout(0.3))
model.add(layers.Dense(10,activation='softmax'))

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

history=model.fit(x_train, y_train, epochs=10, batch_size=20,validation_data=(val_x, val_y))
```

```
plt.plot(history.history["accuracy"])
plt.plot(history.history['val_accuracy'])
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title("model accuracy")
plt.ylabel("Accuracy")
plt.xlabel("Epoch")
plt.legend(["Accuracy","Validation Accuracy","loss","Validation Loss"])
plt.show()
```



To improve our model we made a comparatively complex model by increasing the parameters. We added more convolution layers, increase the kernel size and nods number. We also regularize by adding Dropout and Batch normalization. We got improved accuracy to the test data.

```
test_loss, test_acc = model.evaluate(x_test, y_test)
```

```
313/313 [==============================] - 2s 8ms/step - loss: 1.0035 - accuracy: 0.6710
```

```
test_acc
```

```
0.6710000038146973
```

```
test_loss
```

```
1.003515601158142
```

2 Transfer learning using VGG16

1) Loading the vgg16 model

```
from keras.applications.vgg16 import VGG16
# The convolutional base
vgg_conv = VGG16(weights = "imagenet",include_top = False,input_shape=(32, 32, 3))
```

2)

```
model = models.Sequential()
model.add(vgg_conv)
model.add(layers.Flatten())
model.add(layers.Dense(64,activation='relu'))
model.add(layers.Dense(10,activation='softmax'))
model.summary()
```

```
## Model: "sequential_2"
## _________________________________________________________________
##  Layer (type)                Output Shape              Param #
## =================================================================
##  vgg16 (Functional)          (None, 1, 1, 512)         14714688
##
##  flatten_2 (Flatten)         (None, 512)               0
##
```

```
##  dense_4 (Dense)             (None, 64)                32832
##
##  dense_5 (Dense)             (None, 10)                650
##
## =================================================================
## Total params: 14,748,170
## Trainable params: 14,748,170
## Non-trainable params: 0
## _________________________________________________________________
```

In the dense top layer we have 32832 parameters. In the previous part CNN(task 2.4) there were 65600 parameters in the dense top layer.

3)

Freezing convolutional base

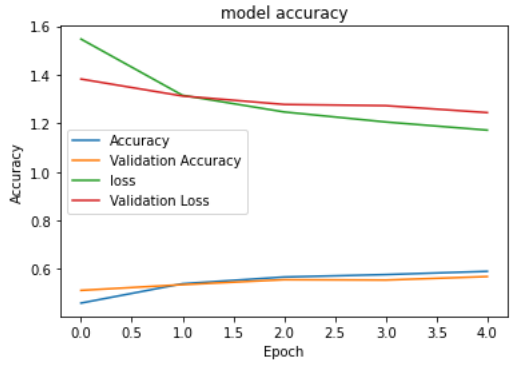
```
vgg_conv = VGG16(weights = "imagenet",include_top = False,input_shape=(32, 32, 3))
vgg_conv.trainable = False
```

```
model = models.Sequential()
model.add(vgg_conv)
model.add(layers.Flatten())
model.add(layers.Dense(64,activation='relu'))
model.add(layers.Dense(10,activation='softmax'))

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

history=model.fit(x_train, y_train, epochs=5, batch_size=16,validation_data=(val_x, val_y))
```

```
plt.plot(history.history["accuracy"])
plt.plot(history.history['val_accuracy'])
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title("model accuracy")
plt.ylabel("Accuracy")
plt.xlabel("Epoch")
plt.legend(["Accuracy","Validation Accuracy","loss","Validation Loss"])
plt.show()
```



After 2 epochs the difference between accuracy and validation accuracy increased but not very over fitted.

```
test_loss, test_acc = model.evaluate(x_test, y_test)
```

```
313/313 [==============================] - 6s 20ms/step - loss: 1.2752 - accuracy: 0.5576
0.5576000213623047
```

```
test_acc
```

```
0.5576000213623047
```

```
test_loss
```

```
1.2751790285110474
```

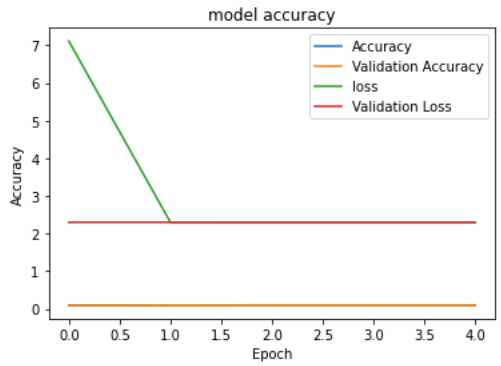
```
vgg_conv = VGG16(weights = "imagenet",include_top = False,input_shape=(32, 32, 3))
vgg_conv.trainable = TRUE
```

```
model = models.Sequential()
model.add(vgg_conv)
model.add(layers.Flatten())
model.add(layers.Dense(64,activation='relu'))
model.add(layers.Dense(10,activation='softmax'))

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

history=model.fit(x_train, y_train, epochs=5, batch_size=16,validation_data=(val_x, val_y))
```

```
plt.plot(history.history["accuracy"])
plt.plot(history.history['val_accuracy'])
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title("model accuracy")
plt.ylabel("Accuracy")
plt.xlabel("Epoch")
plt.legend(["Accuracy","Validation Accuracy","loss","Validation Loss"])
plt.show()
```



After first epochs accuracy and validation accuracy became the same number and unchanged.

```
test_loss, test_acc = model.evaluate(x_test, y_test)
```

```
313/313 [==============================] - 6s 19ms/step - loss: 2.3028 - accuracy: 0.1000
0.10000000149011612
```

```
test_acc
```

```
0.10000000149011612
```

```
test_loss
```

```
2.3027689456939697
```

4)

With freezing the VGG16 convolutional base we get 55.8% accuracy. In the previous model(test 2.4) the accuracy was also 56.5%. In both cases we get almost same result. But when we unfreeze the convolutional base we got a very poor result. The accuracy for the test data was only 10%.