

ML_Cifar100_fin_fin-1101124_fin-

November 24, 2019

1 Final project CIFAR100

-- Network used VGG16

- **Pretrained VGG model (using weights of imageNet)**

Student Name/ID : Aloukik aditya(115290) , Sarthak Rawat(1101124)

```
[1]: from __future__ import absolute_import, division, print_function, \
      → unicode_literals

import tensorflow as tf
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.layers import Flatten, Dense, \
      → Dropout, concatenate, Activation
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras import optimizers
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.callbacks import CSVLogger
from sklearn.preprocessing import OneHotEncoder
from tensorflow.keras.datasets import cifar100
import timeit
import cv2
import gc
import random
from tensorflow.python.keras.callbacks import TensorBoard
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.regularizers import l1
from tensorflow.keras.callbacks import LearningRateScheduler

print("Setup Done..")
```

Setup Done..

1.1 Loading test and train data

- We use `cifar10.load_data()` to load our data

```
[2]: (train_imgs, train_labels), (test_imgs, test_labels) = cifar100.load_data()
print("Data loaded")
```

Data loaded

1.2 Checking data shape and size

```
[3]: print("Shape of train data: ",train_imgs.shape)
print("Shape of test data: ",test_imgs.shape)
print("Shape of train labels: ",train_labels.shape)
print("Shape of test labels: ",test_labels.shape)
```

Shape of train data: (50000, 32, 32, 3)

Shape of test data: (10000, 32, 32, 3)

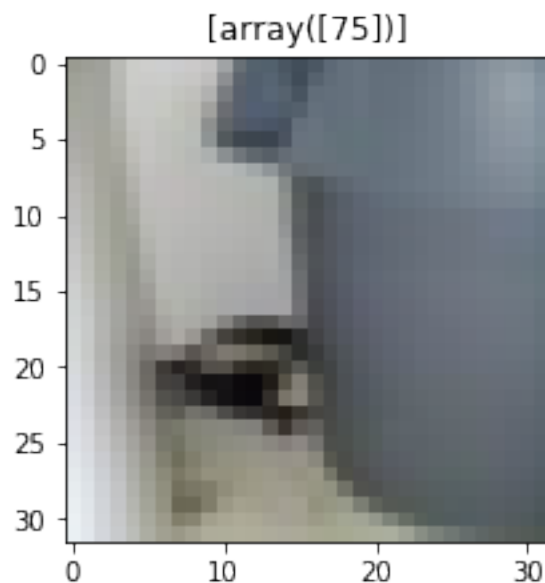
Shape of train labels: (50000, 1)

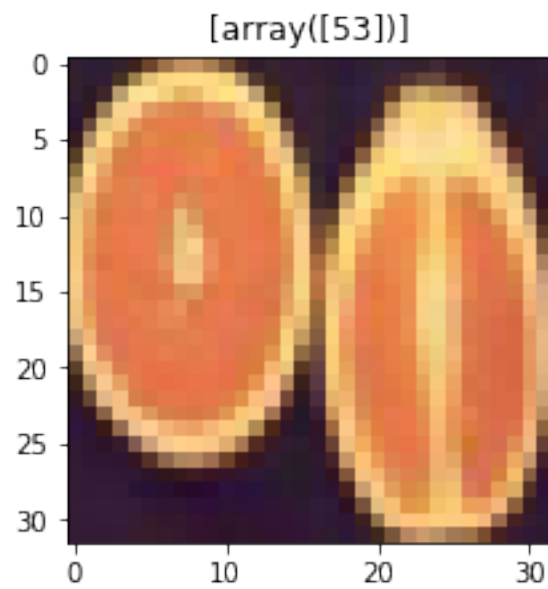
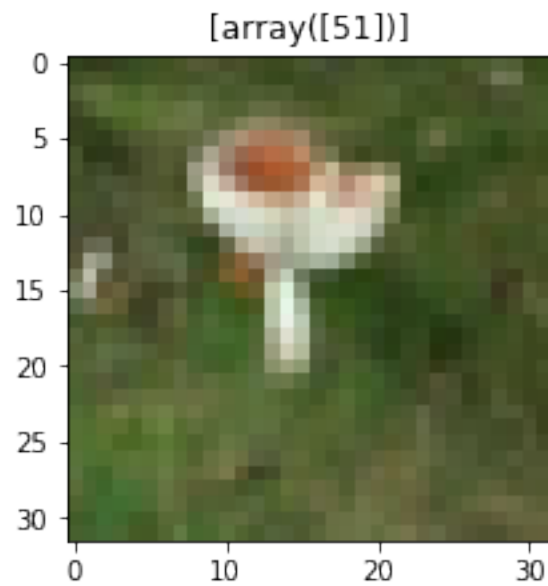
Shape of test labels: (10000, 1)

1.3 Plotting some random images

```
[4]: val_index = []
for i in range(3):
    value = random.randint(0,3582)
    val_index.append(value)
plt.figure( figsize=(15, 15))
plt.subplot(1, 4, i+1)
plt.title([train_labels[value]])
plt.imshow(train_imgs[value])

plt.show()
```





2 Preprocess image using VGG function image

- The function is used to normalize image according to VGG16 network

```
[5]: from tensorflow.keras.applications.vgg16 import preprocess_input
start = timeit.default_timer()
train_imgs = preprocess_input(train_imgs)
test_imgs = preprocess_input(test_imgs)
stop = timeit.default_timer()
print('Time taken to preprocess/normalize data : ', stop - start)
```

Time taken to preprocess/normalize data : 0.5525247910000104

2.1 Resize test and train images

- The images are resized to (224,224,3) format so that they can be used with VGG model
- The below code takes up lot of RAM so be sure to increase your ram on google cloud (otherwise you may get memory error)

```
[6]: gc.collect()
start = timeit.default_timer()
train_imgs = np.array([cv2.resize(img, (224, 224), interpolation=cv2.
    ↳INTER_CUBIC) for img in train_imgs])
test_imgs = np.array([cv2.resize(img, (224, 224), interpolation=cv2.INTER_CUBIC)
    ↳for img in test_imgs])
stop = timeit.default_timer()
print('Time taken to resize data : ', stop - start)
print("Resize done")
```

Time taken to resize data : 44.958423479000004

Resize done

```
[7]: #check shape of newly resized image
train_imgs.shape
```

```
[7]: (50000, 224, 224, 3)
```

2.2 Applying one hot encoding on our labels

```
[8]: onehot_encoder = OneHotEncoder(sparse=False)

train_labels = train_labels.reshape(-1,1)
test_labels = test_labels.reshape(-1,1)

train_labels = onehot_encoder.fit_transform(train_labels)
test_labels = onehot_encoder.transform(test_labels)
```

/usr/local/lib/python3.5/dist-packages/sklearn/preprocessing/_encoders.py:415:
FutureWarning: The handling of integer data will change in version 0.22.
Currently, the categories are determined based on the range [0, max(values)],
while in the future they will be determined based on the unique values.

If you want the future behaviour and silence this warning, you can specify "categories='auto'".

In case you used a LabelEncoder before this OneHotEncoder to convert the categories to integers, then you can now use the OneHotEncoder directly.

```
warnings.warn(msg, FutureWarning)
```

```
[9]: print("Shape of train labels: ",train_labels.shape)
      print("Shape of test labels: ",test_labels.shape)
```

```
Shape of train labels: (50000, 100)
```

```
Shape of test labels: (10000, 100)
```

2.3 Using model from keras--> VGG16

- we have used pretrained weights for imagenet and kept the top fully connected layers.

2.4 How we train our model:

- we freeze our layers of VGG16 so that they are not trained initially
- we only train our new Fully connected layers (so that they start extracting meanings from VGG16 network)
- we only use 2 epochs with high learning rate
- we then unfreeze VGG16 layers, and then train our model with a low variable learning rate for higher number of epoch.

2.4.1 Fully connected layers(Description):

- We use 2 fully connected layers and then add a drop out layer in between to reduce overfitting.
- We also add Regularizer to prevent overfitting
- we have used Adam optimizer with learning rate of 0.0001 for training FC layers for 2 epochs
- for later training of full model we have used variable Learning rates

2.4.2 Freezing VGG16 layers

- we freeze vgg16 model for our first 2 epoch (only new fully connected layer is trained)
- after these epoch we unfreeze the model and train it

```
[11]: #define variable learning rates

def learning_rate_schedule(epoch):
    if epoch <= 3:
        return 1e-4 # 0.00001
    elif epoch <= 8:
        return 1e-5
    elif epoch <= 10:
        return 1e-6
    else:
        return 1e-7
```

```
return LR
```

```
[12]: #Inititalize vgg net
vgg_net = VGG16(include_top=True, weights='imagenet', input_shape=(224,224,3))

#Define our model
x = vgg_net.get_layer('fc2').output
x = Dense(1000, activity_regularizer=l1(0.001), activation='relu',
        name='my_fc3')(x)
x = Dropout(0.3)(x)
x = Dense(100, activation='softmax', name='predictions')(x)
model_updated = Model(inputs=vgg_net.input, outputs=x)

## save our model weights
model_updated.save_weights('CIFAR100_VGG16_model_updated_initial.h5')
print("Model_Updated saved")

history_fc_list = []
history_full_list = []
```

WARNING:tensorflow:From /usr/local/lib/python3.5/dist-packages/tensorflow_core/python/ops/resource_variable_ops.py:1630: calling BaseResourceVariable.__init__ (from tensorflow.python.ops.resource_variable_ops) with constraint is deprecated and will be removed in a future version.
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
Model_Updated saved

2.5 Run 1

```
[13]: i=0
#Inititalize vgg net
vgg_net = VGG16(include_top=True, weights='imagenet', input_shape=(224,224,3))

#Define our model
x = vgg_net.get_layer('fc2').output
x = Dense(1000, activity_regularizer=l1(0.001), activation='relu',
        name='my_fc3')(x)
x = Dropout(0.3)(x)
x = Dense(100, activation='softmax', name='predictions')(x)
model_updated = Model(inputs=vgg_net.input, outputs=x)

gc.collect()

model_updated.load_weights('CIFAR100_VGG16_model_updated_initial.h5')
```

```

print('-----New run started-----')
print('Initial model weights loaded , Started training run number ', i+1)
print('-----')

# Make the last Fully connected layers trainable and freezing the rest of VGG
→model
for layer in model_updated.layers:
    layer.trainable = False
for layer in model_updated.layers[-3:]:
    layer.trainable = True

tensorboard = TensorBoard(log_dir="logs\{}".format('CIFAR100\FC_Layer'+str(i)))

model_updated.compile(loss='categorical_crossentropy', optimizer=optimizers.
→Adam(lr=0.0001), metrics=['accuracy'])

print('-----Starting Fully connected layer
→training-----')

history_fc = model_updated.fit(train_imgs, train_labels, batch_size=128,\
shuffle=True, epochs=2, validation_data=\
(test_imgs, test_labels),callbacks = [tensorboard])

gc.collect()
##Unfreeze weights ----- Now full model will be trained
for layer in model_updated.layers:
    layer.trainable = True

no_epochs = 10
tensorboard = TensorBoard(log_dir="logs\{}".format('CIFAR100\FULL_Model'+str(i)))

opt2 = optimizers.Adam(lr=0.00001)

### Variable learning rate -----
lrate = LearningRateScheduler(learning_rate_schedule)
callbacks_list = [lrate,tensorboard]

model_updated.compile(loss='categorical_crossentropy', optimizer=opt2,
→metrics=['accuracy'])

```

```

print('-----Starting Full model training -->after_
→unfreez-----')

## Training full model
history = model_updated.fit(train_imgs, train_labels, batch_size=128,\
shuffle=True, epochs=no_epochs, validation_data=\
(test_imgs, test_labels),callbacks =callbacks_list)

## Timer for checking time taken
stop = timeit.default_timer()
print('Time taken for one run is : ', stop - start)

model_updated.save_weights('CIFAR100_VGG16_model_updated_fin.h5')
print("Model_Updated weights saved")

## saving history for plots
history_fc_list.append(history_fc)
history_full_list.append(history)

```

```

-----New run started-----
Initial model weights loaded , Started training run number 1
-----

-----Starting Fully connected layer
training-----
Train on 50000 samples, validate on 10000 samples
Epoch 1/2
50000/50000 [=====] - 156s 3ms/sample - loss: 2.4958 -
acc: 0.4627 - val_loss: 1.7918 - val_acc: 0.6034
Epoch 2/2
50000/50000 [=====] - 148s 3ms/sample - loss: 1.6644 -
acc: 0.6300 - val_loss: 1.6049 - val_acc: 0.6420
-----Starting Full model training -->after
unfreez-----
Train on 50000 samples, validate on 10000 samples
Epoch 1/10
50000/50000 [=====] - 373s 7ms/sample - loss: 1.6088 -
acc: 0.6248 - val_loss: 1.3446 - val_acc: 0.6769
Epoch 2/10
50000/50000 [=====] - 368s 7ms/sample - loss: 0.8589 -
acc: 0.8167 - val_loss: 1.1722 - val_acc: 0.7287
Epoch 3/10
50000/50000 [=====] - 368s 7ms/sample - loss: 0.4383 -
acc: 0.9389 - val_loss: 1.1447 - val_acc: 0.7341
Epoch 4/10

```



```

50000/50000 [=====] - 368s 7ms/sample - loss: 0.2666 -
acc: 0.9831 - val_loss: 1.0702 - val_acc: 0.7581
Epoch 5/10
50000/50000 [=====] - 368s 7ms/sample - loss: 0.1900 -
acc: 0.9974 - val_loss: 0.9557 - val_acc: 0.7902
Epoch 6/10
50000/50000 [=====] - 368s 7ms/sample - loss: 0.1717 -
acc: 0.9992 - val_loss: 0.9575 - val_acc: 0.7947
Epoch 7/10
50000/50000 [=====] - 368s 7ms/sample - loss: 0.1637 -
acc: 0.9994 - val_loss: 0.9636 - val_acc: 0.7963
Epoch 8/10
50000/50000 [=====] - 367s 7ms/sample - loss: 0.1574 -
acc: 0.9996 - val_loss: 0.9787 - val_acc: 0.7970
Epoch 9/10
50000/50000 [=====] - 368s 7ms/sample - loss: 0.1523 -
acc: 0.9995 - val_loss: 0.9959 - val_acc: 0.7979
Epoch 10/10
50000/50000 [=====] - 368s 7ms/sample - loss: 0.1484 -
acc: 0.9997 - val_loss: 1.0052 - val_acc: 0.7970
Time taken for one run is : 4214.904176501
Model_Updated weights saved

```

2.6 Run 2

```

[14]: i=1
      #Inititalize vgg net
      vgg_net = VGG16(include_top=True, weights='imagenet', input_shape=(224,224,3))

      #Define our model
      x = vgg_net.get_layer('fc2').output
      x = Dense(1000, activity_regularizer=l1(0.001), activation='relu',
      ↪name='my_fc3')(x)
      x = Dropout(0.3)(x)
      x = Dense(100, activation='softmax', name='predictions')(x)
      model_updated = Model(inputs=vgg_net.input, outputs=x)

      gc.collect()

      model_updated.load_weights('CIFAR100_VGG16_model_updated_initial.h5')

      print('-----New run started-----')
      print('Initial model weights loaded , Started training run number ', i+1)
      print('-----')

```

```

# Make the last Fully connected layers trainable and freezing the rest of VGG
→model
for layer in model_updated.layers:
    layer.trainable = False
for layer in model_updated.layers[-3:]:
    layer.trainable = True

tensorboard = TensorBoard(log_dir="logs\{}".format('CIFAR100\FC_Layer'+str(i)))

model_updated.compile(loss='categorical_crossentropy', optimizer=optimizers.
→Adam(lr=0.0001), metrics=['accuracy'])

print('-----Starting Fully connected layer
→training-----')

history_fc = model_updated.fit(train_imgs, train_labels, batch_size=128,\
shuffle=True, epochs=2, validation_data=\
(test_imgs, test_labels),callbacks = [tensorboard])

gc.collect()
##Unfreeze weights ----- Now full model will be trained
for layer in model_updated.layers:
    layer.trainable = True

no_epochs = 10
tensorboard = TensorBoard(log_dir="logs\{}".format('CIFAR100\FULL_Model'+str(i)))

opt2 = optimizers.Adam(lr=0.00001)

### Variable learning rate -----
lrate = LearningRateScheduler(learning_rate_schedule)
callbacks_list = [lrate,tensorboard]

model_updated.compile(loss='categorical_crossentropy', optimizer=opt2,
→metrics=['accuracy'])

print('-----Starting Full model training -->after
→unfreez-----')

## Training full model

```

```

history = model_updated.fit(train_imgs, train_labels, batch_size=128,\
shuffle=True, epochs=no_epochs, validation_data=\
(test_imgs, test_labels),callbacks =callbacks_list)

```

```

## Timer for checking time taken

```

```

stop = timeit.default_timer()

```

```

print('Time taken for one run is : ', stop - start)

```

```

model_updated.save_weights('CIFAR100_VGG16_model_updated_fin.h5')

```

```

print("Model_Updated weights saved")

```

```

## saving history for plots

```

```

history_fc_list.append(history_fc)

```

```

history_full_list.append(history)

```

```

-----New run started-----
Initial model weights loaded , Started training run number 2
-----
-----Starting Fully connected layer
training-----
Train on 50000 samples, validate on 10000 samples
Epoch 1/2
50000/50000 [=====] - 146s 3ms/sample - loss: 2.4868 -
acc: 0.4671 - val_loss: 1.8002 - val_acc: 0.5989
Epoch 2/2
50000/50000 [=====] - 145s 3ms/sample - loss: 1.6657 -
acc: 0.6291 - val_loss: 1.6197 - val_acc: 0.6356
-----Starting Full model training -->after
unfreez-----
Train on 50000 samples, validate on 10000 samples
Epoch 1/10
50000/50000 [=====] - 414s 8ms/sample - loss: 1.7075 -
acc: 0.6022 - val_loss: 1.2766 - val_acc: 0.7015
Epoch 2/10
50000/50000 [=====] - 413s 8ms/sample - loss: 0.8658 -
acc: 0.8151 - val_loss: 1.1918 - val_acc: 0.7228
Epoch 3/10
50000/50000 [=====] - 413s 8ms/sample - loss: 0.4430 -
acc: 0.9374 - val_loss: 1.1533 - val_acc: 0.7366
Epoch 4/10
50000/50000 [=====] - 413s 8ms/sample - loss: 0.2742 -
acc: 0.9810 - val_loss: 1.0849 - val_acc: 0.7550
Epoch 5/10
50000/50000 [=====] - 413s 8ms/sample - loss: 0.1949 -
acc: 0.9973 - val_loss: 0.9533 - val_acc: 0.7894

```

```

Epoch 6/10
50000/50000 [=====] - 413s 8ms/sample - loss: 0.1738 -
acc: 0.9992 - val_loss: 0.9586 - val_acc: 0.7936
Epoch 7/10
50000/50000 [=====] - 413s 8ms/sample - loss: 0.1649 -
acc: 0.9994 - val_loss: 0.9724 - val_acc: 0.7953
Epoch 8/10
50000/50000 [=====] - 412s 8ms/sample - loss: 0.1586 -
acc: 0.9996 - val_loss: 0.9812 - val_acc: 0.7962
Epoch 9/10
50000/50000 [=====] - 412s 8ms/sample - loss: 0.1531 -
acc: 0.9996 - val_loss: 0.9968 - val_acc: 0.7964
Epoch 10/10
50000/50000 [=====] - 412s 8ms/sample - loss: 0.1490 -
acc: 0.9996 - val_loss: 1.0084 - val_acc: 0.7970
Time taken for one run is : 8780.234548469
Model_Updated weights saved

```

2.7 Run 3

```

[22]: # Function to free up keras memory

from keras.backend.tensorflow_backend import set_session
from keras.backend.tensorflow_backend import clear_session
from keras.backend.tensorflow_backend import get_session
import tensorflow

# Reset Keras Session
def reset_keras():
    sess = get_session()
    clear_session()
    sess.close()
    sess = get_session()

    try:
        del classifier # this is from global space - change this as you need
    except:
        pass

    print(gc.collect()) # if it's done something you should see a number being
    ↳outputted

    # use the same config as you used to create the session
    config = tensorflow.ConfigProto()
    config.gpu_options.per_process_gpu_memory_fraction = 1
    config.gpu_options.visible_device_list = "0"
    set_session(tensorflow.Session(config=config))

```

```
reset_keras()
```

60

```
[23]: i=2
      #Initilize vgg net
      vgg_net = VGG16(include_top=True, weights='imagenet', input_shape=(224,224,3))

      #Define our model
      x = vgg_net.get_layer('fc2').output
      x = Dense(1000, activity_regularizer=l1(0.001), activation='relu',
      ↪name='my_fc3')(x)
      x = Dropout(0.3)(x)
      x = Dense(100, activation='softmax', name='predictions')(x)
      model_updated = Model(inputs=vgg_net.input, outputs=x)

      gc.collect()

      model_updated.load_weights('CIFAR100_VGG16_model_updated_initial.h5')

      print('-----New run started-----')
      print('Initial model weights loaded , Started training run number ', i+1)
      print('-----')

      # Make the last Fully connected layers trainable and freezing the rest of VGG
      ↪model
      for layer in model_updated.layers:
          layer.trainable = False
      for layer in model_updated.layers[-3:]:
          layer.trainable = True

      tensorboard = TensorBoard(log_dir="logs\\{}".format('CIFAR100\\FC_Layer'+str(i)))

      model_updated.compile(loss='categorical_crossentropy', optimizer=optimizers.
      ↪Adam(lr=0.0001), metrics=['accuracy'])

      print('-----Starting Fully connected layer
      ↪training-----')

      history_fc = model_updated.fit(train_imgs, train_labels, batch_size=128,\
      shuffle=True, epochs=2, validation_data=\\
          (test_imgs, test_labels), callbacks = [tensorboard])
```

```

gc.collect()
##Unfreeze weights ----- Now full model will be trained
for layer in model_updated.layers:
    layer.trainable = True

no_epochs = 10
tensorboard = TensorBoard(log_dir="logs\{}".format('CIFAR100\FULL_Model'+str(i)))

opt2 = optimizers.Adam(lr=0.00001)

### Variable learning rate -----
lr_scheduler = LearningRateScheduler(learning_rate_scheduler)
callbacks_list = [lr_scheduler, tensorboard]

model_updated.compile(loss='categorical_crossentropy', optimizer=opt2,
    ↳metrics=['accuracy'])

print('-----Starting Full model training -->after_
    ↳unfreez-----')

## Training full model
history = model_updated.fit(train_imgs, train_labels, batch_size=128,
    shuffle=True, epochs=no_epochs, validation_data=
        (test_imgs, test_labels), callbacks = callbacks_list)

## Timer for checking time taken
stop = timeit.default_timer()
print('Time taken for one run is : ', stop - start)

model_updated.save_weights('CIFAR100_VGG16_model_updated_fin.h5')
print("Model_Updated weights saved")

## saving history for plots
history_fc_list.append(history_fc)
history_full_list.append(history)

```

```

-----New run started-----
Initial model weights loaded , Started training run number 3
-----
-----Starting Fully connected layer

```

```

training-----
Train on 50000 samples, validate on 10000 samples
Epoch 1/2
50000/50000 [=====] - 148s 3ms/sample - loss: 2.5004 -
acc: 0.4642 - val_loss: 1.7935 - val_acc: 0.6043
Epoch 2/2
50000/50000 [=====] - 148s 3ms/sample - loss: 1.6611 -
acc: 0.6307 - val_loss: 1.6149 - val_acc: 0.6367
-----Starting Full model training -->after
unfreez-----
Train on 50000 samples, validate on 10000 samples
Epoch 1/10
50000/50000 [=====] - 368s 7ms/sample - loss: 1.7926 -
acc: 0.5760 - val_loss: 1.3295 - val_acc: 0.6864
Epoch 2/10
50000/50000 [=====] - 367s 7ms/sample - loss: 0.8948 -
acc: 0.8058 - val_loss: 1.1286 - val_acc: 0.7399
Epoch 3/10
50000/50000 [=====] - 367s 7ms/sample - loss: 0.4551 -
acc: 0.9333 - val_loss: 1.1656 - val_acc: 0.7277
Epoch 4/10
50000/50000 [=====] - 367s 7ms/sample - loss: 0.2659 -
acc: 0.9837 - val_loss: 1.0732 - val_acc: 0.7584
Epoch 5/10
50000/50000 [=====] - 367s 7ms/sample - loss: 0.1907 -
acc: 0.9977 - val_loss: 0.9622 - val_acc: 0.7880
Epoch 6/10
50000/50000 [=====] - 367s 7ms/sample - loss: 0.1722 -
acc: 0.9992 - val_loss: 0.9695 - val_acc: 0.7926
Epoch 7/10
50000/50000 [=====] - 367s 7ms/sample - loss: 0.1641 -
acc: 0.9994 - val_loss: 0.9805 - val_acc: 0.7940
Epoch 8/10
50000/50000 [=====] - 367s 7ms/sample - loss: 0.1580 -
acc: 0.9996 - val_loss: 0.9918 - val_acc: 0.7938
Epoch 9/10
50000/50000 [=====] - 367s 7ms/sample - loss: 0.1527 -
acc: 0.9995 - val_loss: 1.0068 - val_acc: 0.7959
Epoch 10/10
50000/50000 [=====] - 367s 7ms/sample - loss: 0.1488 -
acc: 0.9997 - val_loss: 1.0143 - val_acc: 0.7968
Time taken for one run is : 13945.319971805
Model_Updated weights saved

```

2.8 Tensorboard logs have been save to logs folder

- where "_Layer" folder means logs where we train only our FC layers
- where "_Model" folder means logs where we train all layers in our model

2.9 Plotting the graphs of testing accuracy

- Test accuracy
- Plotted for only FC_layer training
- Plotted for full model training

```
[29]: plt.subplot(1, 3, 1)
plt.plot(history_full_list[0].history['val_acc'])
plt.legend(['test'], loc='lower right')

plt.subplot(1, 3, 2)
plt.plot(history_full_list[1].history['val_acc'])
plt.legend(['test'], loc='lower right')

plt.subplot(1, 3, 3)
plt.plot(history_full_list[2].history['val_acc'])
plt.legend(['test'], loc='lower right')
plt.tight_layout()

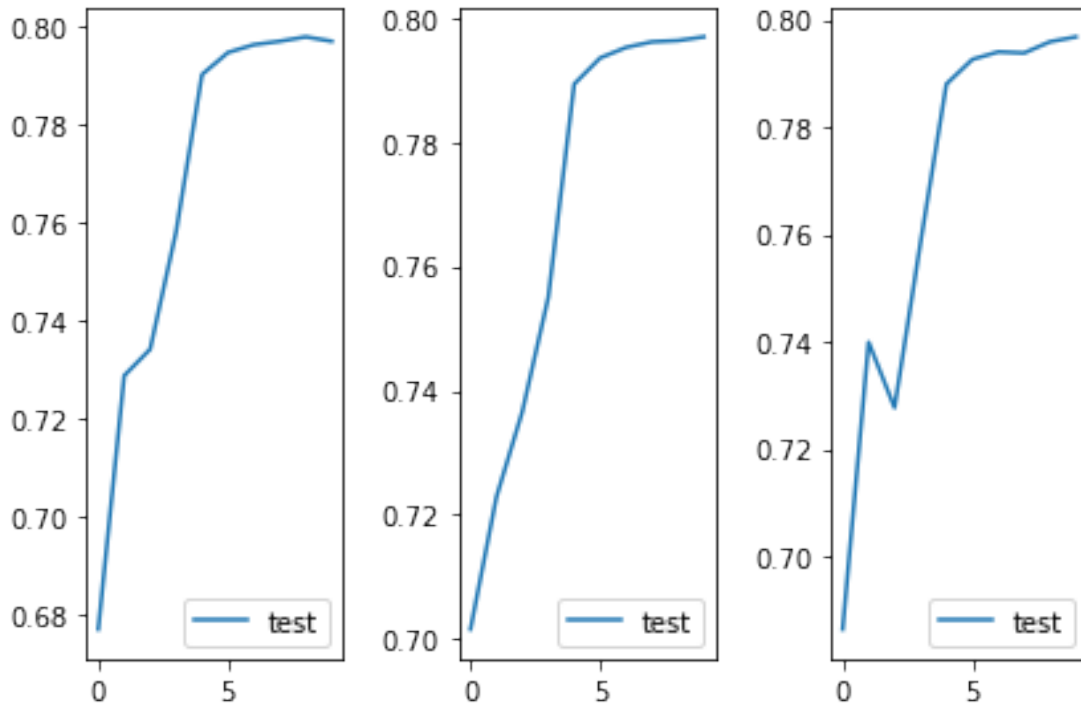
plt.show()

avg_train_acc = 0
avg_test_acc = 0
final_train_loss = []
final_test_loss = []

for his in history_full_list:
    final_train_loss.append(his.history['loss'][-1])
    final_test_loss.append(his.history['val_loss'][-1])
    avg_train_acc = avg_train_acc + his.history['acc'][-1]
    avg_test_acc = avg_test_acc + his.history['val_acc'][-1]

avg_train_acc = avg_train_acc/len(history_fc_list)
avg_test_acc = avg_test_acc/len(history_fc_list)

print("Average testing accuracy: {}".format(avg_test_acc))
```

Average testing accuracy: 0.7969333330790201

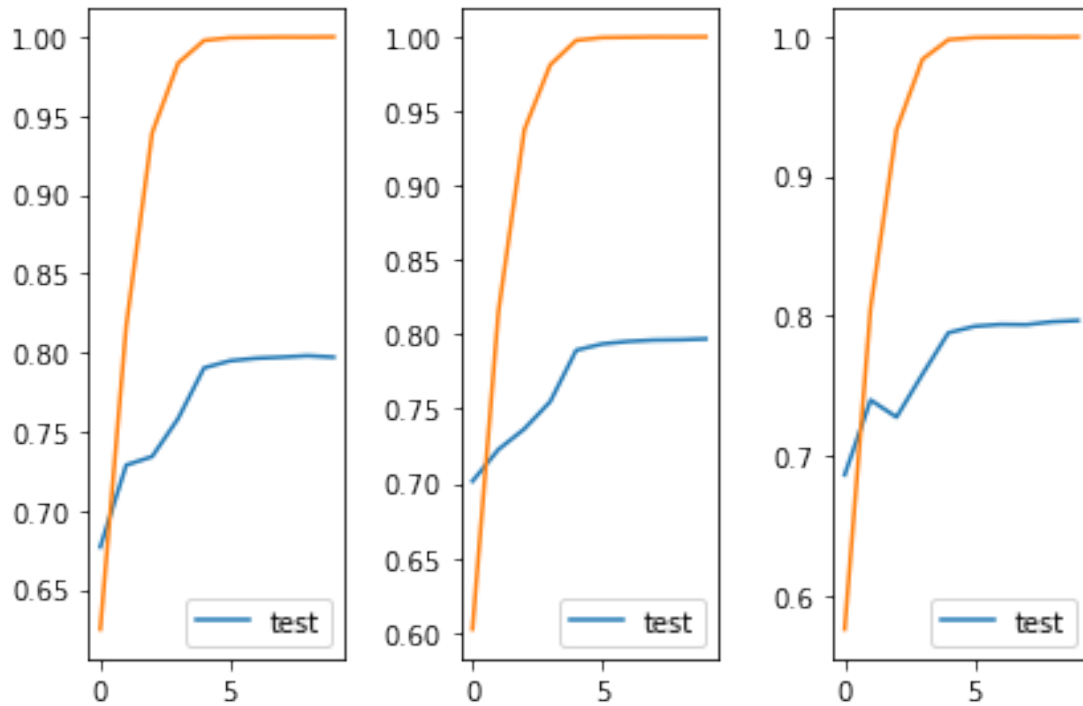
2.10 Train acc vs Test acc

```
[31]: plt.subplot(1, 3, 1)
plt.plot(history_full_list[0].history['val_acc'])
plt.plot(history_full_list[0].history['acc'])
plt.legend(['test'], loc='lower right')

plt.subplot(1, 3, 2)
plt.plot(history_full_list[1].history['val_acc'])
plt.plot(history_full_list[1].history['acc'])
plt.legend(['test'], loc='lower right')

plt.subplot(1, 3, 3)
plt.plot(history_full_list[2].history['val_acc'])
plt.plot(history_full_list[2].history['acc'])
plt.legend(['test'], loc='lower right')
plt.tight_layout()

plt.show()
print("Average train accuracy: {}".format(avg_train_acc))
```



Average train accuracy: 0.999673326810201

3 Final model accuracy is :

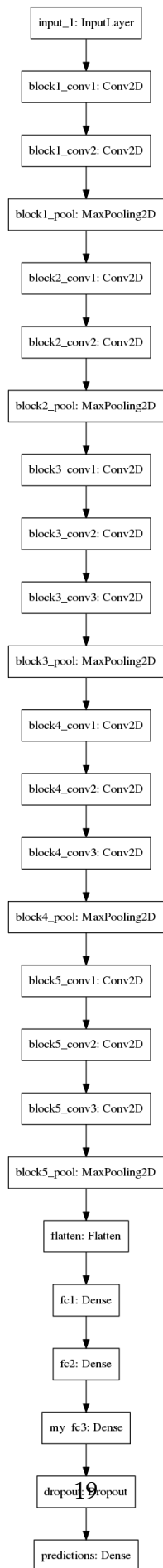
```
[32]: print("Average Model Train accuracy is : ",avg_test_acc*100,"%")
```

Average Model Train accuracy is : 79.69333330790201 %

3.0.1 Model structure is saved to an image.

```
[33]: from tensorflow.keras.utils import plot_model
plot_model(model_updated, to_file='model_CIFAR100_VGG16.png')
```

[33]:



Thank you