> # Name: *Omar Rashad Salem*
>
> # Course: *CV - prof.Heba*
>
> # Assignemnt No.: *6*

> ## QUESTIONS

*1)* **Explain the purpose of Batch Normalization in deep neural networks?**

**ans:** Batch norm. layers help imporving model overall performance by specifically boosting training stability and accelerating training process.

---

*2)* **Describe the two main steps in the Batch Normalization process: normalization and scale/shift?**

**ans:**

- 1. normalization: trying to get each batch `mean` to `0` and `variance` to `1` by subtracting the mean from the batch and deviding by `STD`

$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$

$${\sigma_j}^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{i,j} - \mu_j)^2$$

$$\therefore \hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{{\sigma_j}^2 + \epsilon}}(N \times D)$$

- 2. scale\shift: introducing new 2 learnable hyperparameters `gamma` and `beta` that finds the optimal scale and shift value of the normalized batch which makes the model converge faster

$$y = \gamma\hat{x} + \beta$$

---

*3)* **How do the learnable parameters, gamma and beta, contribute to Batch Normalization?**

**ans:** Gamma $\gamma$ and beta $\beta$ are parameters that are learned during training through backpropagation. These parameters enable the model to decide the optimal scale and shift for the normalized data.

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

---

**4) What was the groundbreaking contribution of AlexNet to the field of Convolutional NeuralNetworks?**

**ans:**

- Introduced one of the first usefull Deep not shallow ConvNet models (Achieved best scores in 2012 ImageNet Classification Challenge error = `16.4%`)
- Was one of the first models to use ReLU nonlinearities
- Itroduced very early normilzation method called `local response normlization`

---

**5) How does the VGG architecture differ from other ConvNet architectures in terms of filter size?**

**ans:** Introduced a fixed rules to use:

- only `3x3 s=1 p=1` conv filters making scalling the model much easier
- only max pooling layers `2x2 s=2` filters

---

**6) Explain the concept of the inception module in GoogLeNet and its advantages?**

**ans:**

- Inception module is a repeated local structure in GoogLeNet arch. works by simultaneously calculating multiple/all conv kernel sizes in same level then concatenate them effectively eleminating the need for kernel/filter size as a hyperparameter.

- They also used bottleneck layer (filters) to reduce feature map dimensions before any expensive computation layers.

---

**7) What is the main innovation introduced by ResNet to address training challenges in deepnetworks?**

**ans:**

- Before ResNets after specific deep/depth ConvNets starts performing worse!
- ResNets came to solve this issue by making learning the identity functions with high depth models easy and now deeper models again behaves as expeceted (better than shallower models)
- After ResNet now we have two main blocks in Convnets a `Plain block` and a `Residual block`

---

> # Programming assignment:

In [ ]:

```
# Omar rashad note: un-comment next line if first time using keras
#! pip install keras

from sklearn.utils.multiclass import unique_labels
from sklearn.model_selection import train_test_split
from keras import Sequential
from keras.applications import VGG16, ResNet50
from keras.optimizers import SGD
from keras.layers import Flatten,Dense #MAY NEED: BatchNormalization,Activation,Dropout
from keras.utils import to_categorical



#get the  cifar10
from keras.datasets import cifar10
(x_train,y_train),(x_test,y_test)=cifar10.load_data()

#get validation subset
x_train,x_val,y_train,y_val=train_test_split(x_train,y_train,test_size=.3)

#turn to onehot
y_train=to_categorical(y_train)
y_val=to_categorical(y_val)
y_test=to_categorical(y_test)

#Defining the VGG and ResNet50
vgg16_base_model = VGG16(include_top=False,weights='imagenet',input_shape=(32,32,3),classes=y_train.shape[1])
resnet50_base_model = ResNet50(include_top=False,weights='imagenet',input_shape=(32,32,3),classes=y_train.shape[1])

#create our own modified model ( append our dens layer at end)
model= Sequential()
model.add(vgg16_base_model)
model.add(Flatten())

model2= Sequential()
model2.add(resnet50_base_model)
model2.add(Flatten())

#after creating ? yes add the layers!
model.add(Dense(1024,activation=('relu'),input_dim=512))
model.add(Dense(512,activation=('relu')))
model.add(Dense(256,activation=('relu')))
model.add(Dense(128,activation=('relu')))
model.add(Dense(10,activation=('softmax'))) #This is the classification layer

model2.add(Dense(1024,activation=('relu'),input_dim=512))
model2.add(Dense(512,activation=('relu')))
```

```
model2.add(Dense(256,activation=('relu')))
model2.add(Dense(128,activation=('relu')))
model2.add(Dense(10,activation=('softmax'))) #This is the classification layer

#let's see it now! ORS#
print(f'modified VGG model Summary: \n')
model.summary()
print(f'modified ResNet model Summary: \n')
model2.summary()

#hype hype :D hyperparameters
batch_size= 100
epochs=5
learn_rate=.001
sgd=SGD(learning_rate=learn_rate,momentum=.9,nesterov=False)

#all set! COMPILE!
model.compile(optimizer=sgd,loss='categorical_crossentropy',metrics=['accuracy'])
model2.compile(optimizer=sgd,loss='categorical_crossentropy',metrics=['accuracy'])

#train and save train history
print(f'\n\n\n START TRAINING OUR ResNet \n\n\n')
history2 = model2.fit(x_train, y_train, batch_size= batch_size, epochs= epochs, validation_data=(x_val, y_val))
print(f'\n\n\n START TRAINING OUR VGG \n\n\n')
history = model.fit(x_train, y_train, batch_size= batch_size, epochs= epochs, validation_data=(x_val, y_val))
```

```
modified VGG model Summary:

Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 1, 1, 512)         14714688

 flatten_7 (Flatten)         (None, 512)               0

 dense_35 (Dense)            (None, 1024)              525312

 dense_36 (Dense)            (None, 512)               524800

 dense_37 (Dense)            (None, 256)               131328

 dense_38 (Dense)            (None, 128)               32896

 dense_39 (Dense)            (None, 10)                1290

=================================================================
Total params: 15,930,314
Trainable params: 15,930,314
Non-trainable params: 0
_________________________________________________________________
modified ResNet model Summary:

Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50 (Functional)       (None, 1, 1, 2048)        23587712

 flatten_8 (Flatten)         (None, 2048)              0

 dense_40 (Dense)            (None, 1024)              2098176

 dense_41 (Dense)            (None, 512)               524800

 dense_42 (Dense)            (None, 256)               131328

 dense_43 (Dense)            (None, 128)               32896

 dense_44 (Dense)            (None, 10)                1290

=================================================================
Total params: 26,376,202
Trainable params: 26,323,082
```

```
Non-trainable params: 53,120
_________________________________________________________________
```

In [ ]:
```python
#get the history
print(f'\n\nOur VGG model train history: ')
for info in history.history: print(f'Epochs {info} : {history.history[info]}')

print(f'\n\nOur ResNet model train history: ')
for info in history2.history: print(f'Epochs {info} : {history2.history[info]}')



print('\n\n')
print('Total Train Time for Both models = 88m 30.8s')
print('\n\n')



print('E V A L U A T E')
#evaluate
loss, accuracy = model.evaluate(x_test, y_test)
print(f'Our VGG model final results on cifar10 dataset: \n loss= {loss}\n accuracy= {accuracy}\n\n')
loss2, accuracy2 = model2.evaluate(x_test, y_test)
print(f'Our ResNet model final results on cifar10 dataset: \n loss= {loss2}\n accuracy= {accuracy2}')

#DONE!
```

```
Our VGG model train history:
Epochs loss : [2.331294536590576, 2.302603244781494, 2.302593946456909, 2.3025898933410645, 2.3025882244110107]
Epochs accuracy : [0.09759999811649323, 0.09868571162223816, 0.09757142513990402, 0.09991428256034851, 0.09797143191099167]
Epochs val_loss : [2.3026533126831055, 2.30268931388855, 2.3027286529541016, 2.302746295928955, 2.3027658462524414]
Epochs val_accuracy : [0.09880000352859497, 0.09753333032131195, 0.09753333032131195, 0.09746666997671127, 0.09746666997671127]


Our ResNet model train history:
Epochs loss : [1.592706561088562, 0.8918847441673279, 0.6395752429962158, 0.45397523045539856, 0.342894583940506]
Epochs accuracy : [0.43308570981025696, 0.689542829990387, 0.7785428762435913, 0.8407999873161316, 0.8809428811073303]
Epochs val_loss : [1.126675009727478, 0.8955151438713074, 0.856717050075531, 0.8438029289245605, 0.8935683369636536]
Epochs val_accuracy : [0.6024666428565979, 0.6928666830062866, 0.7077333331108093, 0.73253333568573, 0.7376000285148621]



Total Train Time for Both models = 88m 30.8s



E V A L U A T E
313/313 [==============================] - 18s 57ms/step - loss: 2.3026 - accuracy: 0.1000
Our VGG model final results on cifar10 dataset:
 loss= 2.3026132583618164
 accuracy= 0.10000000149011612


313/313 [==============================] - 15s 48ms/step - loss: 0.8945 - accuracy: 0.7362
Our ResNet model final results on cifar10 dataset:
 loss= 0.894540548324585
 accuracy= 0.7361999750137329
```