Assignment-5

[1]:
```python
import tensorflow as tf

# Display the version
print(tf.__version__)

# other imports
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.layers import Input, Conv2D, Dense, Flatten, Dropout
from tensorflow.keras.layers import GlobalMaxPooling2D, MaxPooling2D
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.models import Model
```

2.15.0

[2]:
```python
# Load in the data
cifar10 = tf.keras.datasets.cifar10

# Distribute it to train and test set
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
print(x_train.shape, y_train.shape, x_test.shape, y_test.shape)
```

(50000, 32, 32, 3) (50000, 1) (10000, 32, 32, 3) (10000, 1)

[3]:
```python
# Reduce pixel values
x_train, x_test = x_train / 255.0, x_test / 255.0

# flatten the label values
y_train, y_test = y_train.flatten(), y_test.flatten()
```

[4]:
```python
# number of classes
K = len(set(y_train))

# calculate total number of classes
# for output layer
print("number of classes:", K)

# Build the model using the functional API
```

```
# input layer
i = Input(shape=x_train[0].shape)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(i)
x = BatchNormalization()(x)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = MaxPooling2D((2, 2))(x)

x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = MaxPooling2D((2, 2))(x)

x = Conv2D(128, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = Conv2D(128, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = MaxPooling2D((2, 2))(x)

x = Flatten()(x)
x = Dropout(0.2)(x)

# Hidden layer
x = Dense(1024, activation='relu')(x)
x = Dropout(0.2)(x)

# last hidden layer i.e.. output layer
x = Dense(K, activation='softmax')(x)

model = Model(i, x)

# model description
model.summary()
```

```
number of classes: 10
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 32, 32, 3)]       0

 conv2d (Conv2D)             (None, 32, 32, 32)        896

 batch_normalization (Batch  (None, 32, 32, 32)        128
 Normalization)
```

conv2d_1 (Conv2D)	(None, 32, 32, 32)	9248
batch_normalization_1 (BatchNormalization)	(None, 32, 32, 32)	128
max_pooling2d (MaxPooling2D)	(None, 16, 16, 32)	0
conv2d_2 (Conv2D)	(None, 16, 16, 64)	18496
batch_normalization_2 (BatchNormalization)	(None, 16, 16, 64)	256
conv2d_3 (Conv2D)	(None, 16, 16, 64)	36928
batch_normalization_3 (BatchNormalization)	(None, 16, 16, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 8, 8, 64)	0
conv2d_4 (Conv2D)	(None, 8, 8, 128)	73856
batch_normalization_4 (BatchNormalization)	(None, 8, 8, 128)	512
conv2d_5 (Conv2D)	(None, 8, 8, 128)	147584
batch_normalization_5 (BatchNormalization)	(None, 8, 8, 128)	512
max_pooling2d_2 (MaxPooling2D)	(None, 4, 4, 128)	0
flatten (Flatten)	(None, 2048)	0
dropout (Dropout)	(None, 2048)	0
dense (Dense)	(None, 1024)	2098176
dropout_1 (Dropout)	(None, 1024)	0
dense_1 (Dense)	(None, 10)	10250

===
Total params: 2397226 (9.14 MB)
Trainable params: 2396330 (9.14 MB)
Non-trainable params: 896 (3.50 KB)

[5]:
```
# Compile
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

[6]:
```
# Fit
r = model.fit(
x_train, y_train, validation_data=(x_test, y_test), epochs=10)
```

```
Epoch 1/10
1563/1563 [==============================] - 20s 8ms/step - loss: 1.3153 -
accuracy: 0.5517 - val_loss: 1.7805 - val_accuracy: 0.4983
Epoch 2/10
1563/1563 [==============================] - 11s 7ms/step - loss: 0.8354 -
accuracy: 0.7099 - val_loss: 1.0786 - val_accuracy: 0.6472
Epoch 3/10
1563/1563 [==============================] - 11s 7ms/step - loss: 0.6803 -
accuracy: 0.7652 - val_loss: 0.7639 - val_accuracy: 0.7428
Epoch 4/10
1563/1563 [==============================] - 12s 8ms/step - loss: 0.5806 -
accuracy: 0.8004 - val_loss: 0.7259 - val_accuracy: 0.7563
Epoch 5/10
1563/1563 [==============================] - 12s 8ms/step - loss: 0.4966 -
accuracy: 0.8290 - val_loss: 0.7507 - val_accuracy: 0.7548
Epoch 6/10
1563/1563 [==============================] - 11s 7ms/step - loss: 0.4200 -
accuracy: 0.8546 - val_loss: 0.6544 - val_accuracy: 0.7887
Epoch 7/10
1563/1563 [==============================] - 18s 11ms/step - loss: 0.3475 -
accuracy: 0.8789 - val_loss: 0.6331 - val_accuracy: 0.8042
Epoch 8/10
1563/1563 [==============================] - 12s 7ms/step - loss: 0.2996 -
accuracy: 0.8971 - val_loss: 0.6709 - val_accuracy: 0.8032
Epoch 9/10
1563/1563 [==============================] - 12s 8ms/step - loss: 0.2601 -
accuracy: 0.9097 - val_loss: 0.6870 - val_accuracy: 0.8097
Epoch 10/10
1563/1563 [==============================] - 11s 7ms/step - loss: 0.2205 -
accuracy: 0.9233 - val_loss: 0.6497 - val_accuracy: 0.8206
```

[7]:
```
# Plot accuracy per iteration
plt.plot(r.history['accuracy'], label='acc', color='red')
plt.plot(r.history['val_accuracy'], label='val_acc', color='green')
plt.legend()
```

[7]: <matplotlib.legend.Legend at 0x7e21f398eb30>