Deep Neural Network Method with Convolution

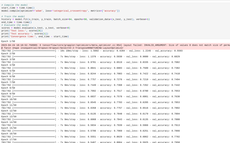
The classification deep learning model has been created with convolutional layers. The data has been split into training and test data. The convolution layers are created using the Conv2D function with ReLU as the activation function and the same padding option. The pooling layers are created using the MaxPooling2D function. The output layer has 10 neurons for the 10 classes and has softmax activation function. The model is compiled using adam optimizer and categorical_crossentropy loss function. The model is fitted using the fit function with 10 epochs. The loss and accuracy of the model are calculated for training and test data. The model is then evaluated by generating the confusion matrix, the classification report and the heatmap plot.

```
# Import the the required libraries
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten, Dense, Dropout
from keras.utils import np_utils
from keras.datasets import cifar10
from keras.preprocessing.image import ImageDataGenerator
from keras.utils import to_categorical
from keras.models import Sequential, Model, load_model
from keras.optimizers import SGD
from keras.callbacks import ReduceLROnPlateau, CSVLogger, EarlyStopping, ModelCheckpoint
from keras.utils.vis_utils import plot_model
from keras.preprocessing.image import load_img, img_to_array
from keras.applications.vgg16 import VGG16, preprocess_input, decode_predictions, VGG16
from keras.applications.mobilenet_v2 import MobileNetV2
from keras.applications.mobilenet_v2 import preprocess_input as mnet_preprocess_input
from keras.applications.mobilenet_v2 import decode_predictions as mnet_decode_predictions
from keras.applications.resnet50 import ResNet50
from keras.applications.resnet50 import preprocess_input as resnet_preprocess_input
from keras.applications.resnet50 import decode_predictions as resnet_decode_predictions
from keras.applications.inception_v3 import InceptionV3
from keras.applications.inception_v3 import preprocess_input as inception_preprocess_input
from keras.applications.inception_v3 import decode_predictions as inception_decode_predictions
from keras.applications.densenet import DenseNet121
from keras.applications.densenet import preprocess_input as densenet_preprocess_input
from keras.applications.densenet import decode_predictions as densenet_decode_predictions
from keras.applications.nasnet import NASNetMobile
from keras.applications.nasnet import preprocess_input as nasnet_preprocess_input
from keras.applications.nasnet import decode_predictions as nasnet_decode_predictions
from keras.applications.efficientnet import EfficientNetB0
from keras.applications.efficientnet import preprocess_input as efficientnet_preprocess_input
from keras.applications.efficientnet import decode_predictions as efficientnet_decode_predictions
from keras.applications.vgg19 import VGG19
from keras.applications.vgg19 import preprocess_input as vgg19_preprocess_input
from keras.applications.vgg19 import decode_predictions as vgg19_decode_predictions
from keras.applications.xception import Xception
from keras.applications.xception import preprocess_input as xception_preprocess_input
from keras.applications.xception import decode_predictions as xception_decode_predictions
from keras.applications.mobilenet import MobileNet
from keras.applications.mobilenet import preprocess_input as mnet_preprocess_input
from keras.applications.mobilenet import decode_predictions as mnet_decode_predictions
%matrix_inline
%tensorflow_version 1.x
```

```
# Load the CIFAR10 data.
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
```

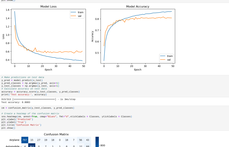
```
Fig, ax= plt.subplots(1,2,figsize=(20,8))
ax[0].set_title('Class Distribution of Training data')
sns.countplot(y_train, palette='husl', ax=ax[0])
ax[0].set_xlabel('Classes')
ax[0].set_ylabel('Count')
ax[1].set_title('Class Distribution of Testing data')
sns.countplot(y_test, palette='husl', ax=ax[1])
ax[1].set_xlabel('Classes')
ax[1].set_ylabel('Count')

# Convert training data to one-hot encoded values.
y_train = to_categorical(y_train, num_classes)
y_test = to_categorical(y_test, num_classes)

# Create a sequential model.
model = Sequential(name="model_cifar10_simple")
model.add(Conv2D(32, (3, 3), padding='same',
                 input_shape=X_train.shape[1:], activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dense(10, activation='softmax'))

# Print model summary.
print(model.summary())

# Compile model.
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

# Fit data to model.
history = model.fit(X_train, y_train,
                    batch_size=32,
                    epochs=10,
                    validation_data=(X_test, y_test),
                    shuffle=True)

# Save neural network structure.
model_structure = model.to_json()
f = Path("model.structure").write_text(model_structure)

# Save neural network parameters.
model.save_weights('model_cifar10_simple.weights.h5')

# Create list of accuracies.
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

# Create list of losses.
loss = history.history['loss']
val_loss = history.history['val_loss']

# Create list of epochs.
epochs = range(1, len(acc) + 1)

# Plot training and validation accuracy.
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

# Plot training and validation loss.
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

# Predict the values from the validation dataset.
Y_pred = model.predict(X_test)
y_pred = np.argmax(Y_pred, axis=1)

# Print classification report.
print("Classification Report")
print(classification_report(y_test, y_pred))

# Print confusion matrix.
print("Confusion Matrix")
print(confusion_matrix(y_test, y_pred))
```

```
Fig, axes= plt.subplots(1,2,figsize=(20,8))
axes[0].imshow(X_train[0])
axes[0].axis('off')
axes[1].imshow(X_test[0])
axes[1].axis('off')
```



```
# Model summary.
Model: "model_cifar10_simple"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 32, 32, 32)        896
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 8, 8, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 4096)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               2097664
_________________________________________________________________
dense_2 (Dense)              (None, 10)                5130
=================================================================
Total params: 2,122,186
Trainable params: 2,122,186
Non-trainable params: 0
_________________________________________________________________
None

# Compile model.
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

# Fit data to model.
history = model.fit(X_train, y_train,
                    batch_size=32,
                    epochs=10,
                    validation_data=(X_test, y_test),
                    shuffle=True)

# Save neural network structure.
model_structure = model.to_json()
f = Path("model.structure").write_text(model_structure)

# Save neural network parameters.
model.save_weights('model_cifar10_simple.weights.h5')

# Create list of accuracies.
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

# Create list of losses.
loss = history.history['loss']
val_loss = history.history['val_loss']

# Create list of epochs.
epochs = range(1, len(acc) + 1)

# Plot training and validation accuracy.
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

# Plot training and validation loss.
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

# Predict the values from the validation dataset.
Y_pred = model.predict(X_test)
y_pred = np.argmax(Y_pred, axis=1)

# Print classification report.
print("Classification Report")
print(classification_report(y_test, y_pred))

# Print confusion matrix.
print("Confusion Matrix")
print(confusion_matrix(y_test, y_pred))
```

```
Epoch 1/10
1563/1563 [==============================] - 7s 4ms/step - loss: 1.6997 - accuracy: 0.3844 - val_loss: 1.3395 - val_accuracy: 0.5080
Epoch 2/10
1563/1563 [==============================] - 6s 4ms/step - loss: 1.2270 - accuracy: 0.5627 - val_loss: 1.1149 - val_accuracy: 0.6070
Epoch 3/10
1563/1563 [==============================] - 6s 4ms/step - loss: 1.0197 - accuracy: 0.6397 - val_loss: 1.0154 - val_accuracy: 0.6447
Epoch 4/10
1563/1563 [==============================] - 6s 4ms/step - loss: 0.8811 - accuracy: 0.6907 - val_loss: 0.9154 - val_accuracy: 0.6827
Epoch 5/10
1563/1563 [==============================] - 6s 4ms/step - loss: 0.7811 - accuracy: 0.7264 - val_loss: 0.8966 - val_accuracy: 0.6917
Epoch 6/10
1563/1563 [==============================] - 6s 4ms/step - loss: 0.6911 - accuracy: 0.7575 - val_loss: 0.8811 - val_accuracy: 0.7000
Epoch 7/10
1563/1563 [==============================] - 6s 4ms/step - loss: 0.6170 - accuracy: 0.7824 - val_loss: 0.7928 - val_accuracy: 0.7252
Epoch 8/10
1563/1563 [==============================] - 6s 4ms/step - loss: 0.5488 - accuracy: 0.8052 - val_loss: 0.8016 - val_accuracy: 0.7253
Epoch 9/10
1563/1563 [==============================] - 6s 4ms/step - loss: 0.4854 - accuracy: 0.8270 - val_loss: 0.8127 - val_accuracy: 0.7264
Epoch 10/10
1563/1563 [==============================] - 6s 4ms/step - loss: 0.4300 - accuracy: 0.8460 - val_loss: 0.8047 - val_accuracy: 0.7361
Classification Report
              precision    recall  f1-score   support

           0       0.77      0.77      0.77      1000
           1       0.87      0.86      0.86      1000
           2       0.64      0.64      0.64      1000
           3       0.54      0.54      0.54      1000
           4       0.69      0.69      0.69      1000
           5       0.64      0.64      0.64      1000
           6       0.78      0.78      0.78      1000
           7       0.76      0.76      0.76      1000
           8       0.83      0.83      0.83      1000
           9       0.79      0.79      0.79      1000

    accuracy                           0.73     10000
   macro avg       0.73      0.73      0.73     10000
weighted avg       0.73      0.73      0.73     10000

Confusion Matrix
[[772  12  61  21  19  11  10  14  71  19]
 [ 11 860   1   2   1   1   6   1  14 103]
 [ 61   1 639  49 103  45  42  21   9   1]
 [ 21   2  49 539  71 163  44  27   7   1]
 [ 19   1 103  71 690  32  29  45   8   2]
 [ 11   1  45 163  32 644  17  42   5   1]
 [ 10   6  42  44  29  17 780   7   4   1]
 [ 14   1  21  27  45  42   7 760   1   2]
 [ 71  14   9   7   8   5   4   1 829  42]
 [ 19 103   1   1   2   1   2   2  42 827]]
```




```
# Plot showing data and predicted.
plt.rcParams['figure.dpi'] = 300
plt.rcParams['font.size'] = 12
plt.rcParams['axes.labelsize'] = 12
plt.rcParams['axes.titlesize'] = 12
plt.rcParams['text'] = 'on'
plt.rcParams['figure.figsize'] = (8,8)
plt.rcParams['image.cmap'] = 'jet'
plt.rcParams['axes.grid'] = False
plt.rcParams['lines.linewidth'] = 1.5
plt.rcParams['lines.markersize'] = 6
plt.rcParams['legend.fontsize'] = 12
plt.rcParams['legend.frameon'] = True
plt.rcParams['legend.shadow'] = True
plt.rcParams['legend.fancybox'] = True
plt.rcParams['legend.numpoints'] = 1
plt.rcParams['legend.scatterpoints'] = 1
plt.rcParams['legend.markerscale'] = 1
plt.rcParams['legend.borderpad'] = 0.4
plt.rcParams['legend.borderaxespad'] = 0.5
plt.rcParams['legend.handlelength'] = 2
plt.rcParams['legend.handleheight'] = 1
plt.rcParams['legend.handletextpad'] = 0.8
plt.rcParams['legend.labelspacing'] = 0.5
plt.rcParams['legend.columnspacing'] = 2
plt.rcParams['legend.loc'] = 'upper right'
plt.rcParams['legend.fancybox'] = True
plt.rcParams['legend.shadow'] = True
plt.rcParams['legend.frameon
```