5/7/2023

CS-351-Artificial Intelligence

2020244 - Mohsin Zia
2020256 – Muhammad Abdullah
2020474 - Syed Irtaza Haider

INSTRUCTOR: PROFESSOR ZAHID HALIM

Cifar 10 – Classification using CNN

- First of all, we import all the required libraries needed in our project.

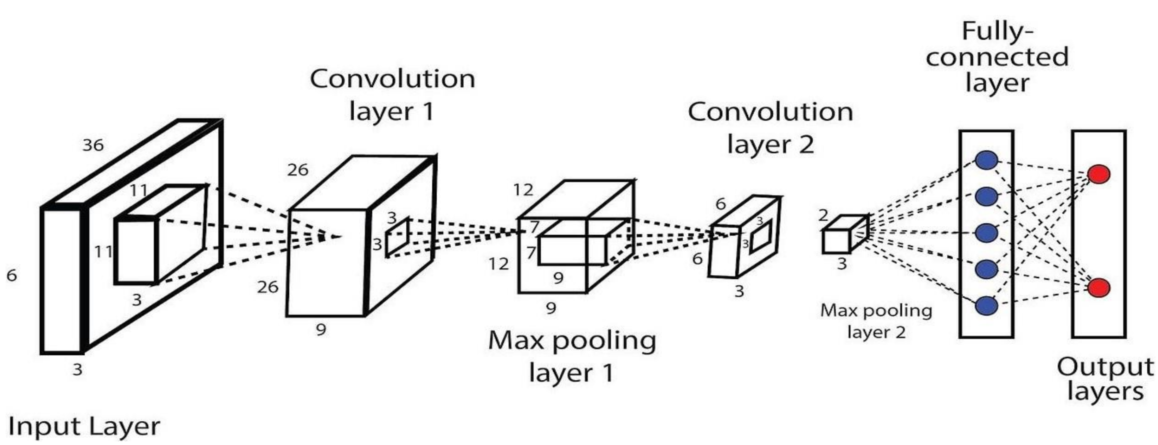
```
import tensorflow as tf
from tensorflow import keras
from keras import datasets, layers, models
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import confusion_matrix
import seaborn as sns
from tensorflow.keras.layers import InputLayer, Conv2D, BatchNormalization, MaxPooling2D, Activation
from tensorflow.keras.layers import Flatten, Dense, Dropout
```

- Then we create a normal CNN architecture **without** using any extra techniques to improve the accuracy of the model:

```
CNN = models.Sequential([
    layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])
```



- Then we compile the model using the **adam** optimizer and **sparse_categorical_crossentropy** as our loss function. We train our model for 20 epochs which results in **89%** accuracy on the train set.

```
CNN.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
CNN.fit(X_train, y_train, epochs=20)
```

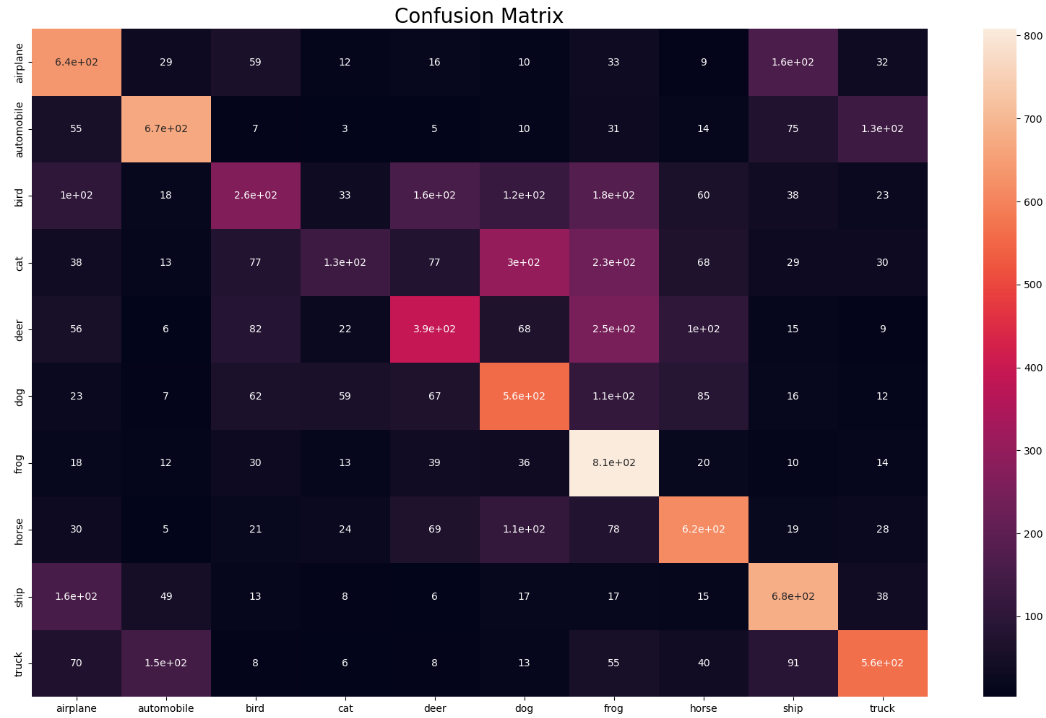
- But then evaluating the model on the test set results in only **68%** accuracy.

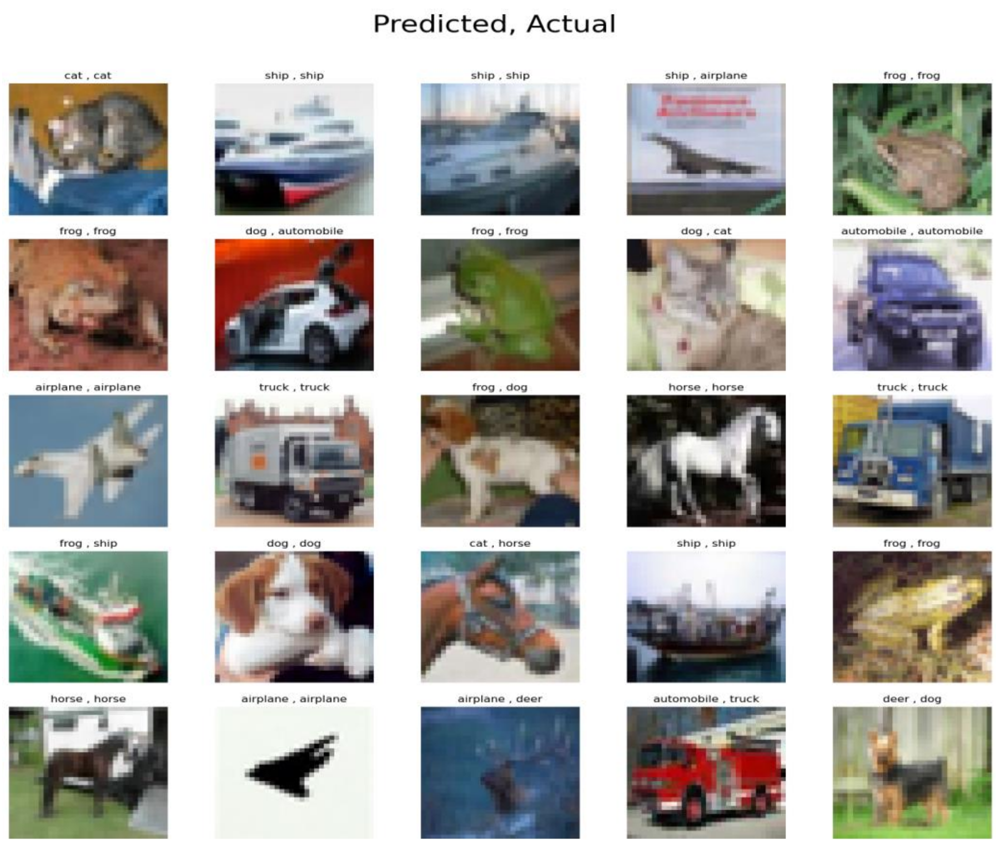
```
CNN.evaluate(X_test,y_test)
labelPredicted = CNN.predict(X_test)
```

- Prediction results of first 10 test set images

```
# Prediction results of 10 test set images images:
Predicted starting 10 labels: [3 8 8 8 6 6 5 6 5 1]
Actual starting 10 labels:    [3 8 8 0 6 6 1 6 3 1]
```

➢ Now we plot the conclusion matrix and comparison of predicted and actual labels.





Now experimenting with different techniques to improve accuracy of our model

- Now first of all we normalize the pixels of our train and test set by subtracting the mean (μ) of each feature and a division by the standard deviation (σ). This way, each feature has a mean of 0 and a standard deviation of 1. This results in faster convergence.

```
X_train = (X_train - X_train.mean()) / X_train.std()
X_test = (X_test - X_test.mean()) / X_test.std()
```

- Then we update our old architecture by adding batch normalization and dropout layers.

```
model.add(Conv2D(filters=16, kernel_size=3, padding='same', activation='elu', use_bias=False))
model.add(BatchNormalization())   # leave default axis=-1 in all BN layers
model.add(MaxPooling2D(pool_size=[2,2], strides=[2, 2], padding='same'))

model.add(Conv2D(filters=32, kernel_size=3, padding='same', activation='elu', use_bias=False))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=[2,2], strides=[2, 2], padding='same'))

model.add(Conv2D(filters=64, kernel_size=3, padding='same', activation='elu', use_bias=False))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=[2,2], strides=[2, 2], padding='same'))

model.add(Dropout(0.3))
model.add(Flatten())
model.add(Dense(512, activation='elu'))
model.add(Dropout(0.3))
model.add(Dense(10, activation='softmax'))
```

- Then we compile the model using the **adam** optimizer and **sparse_categorical_crossentropy** as our loss function. We train our model for 20 epochs while keeping the batch size 256 which results in **95%** accuracy on the train set.

```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(x=X_train, y=y_train_mapped, batch_size=256, epochs=20,
          validation_data=[X_test, y_test_mapped], verbose=2)
```

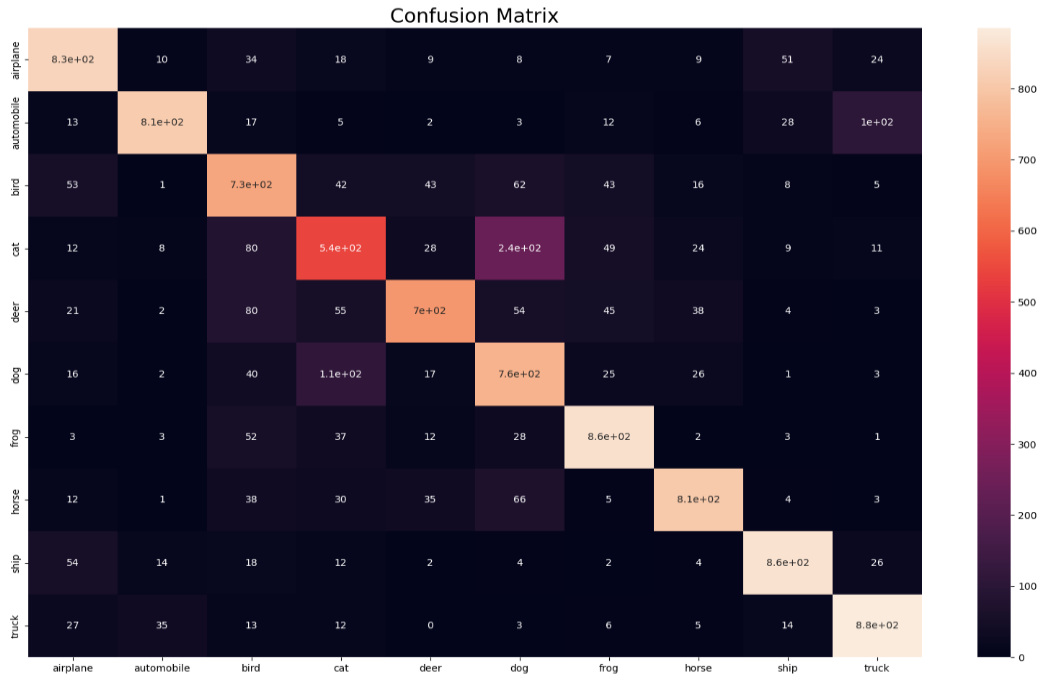
- Then evaluating the model on the test set results in **78%** accuracy.

```
model.evaluate(X_test, y_test_mapped, batch_size=256)
labelPredictedByModel = model.predict(X_test)
```

- Prediction results of first 20 test set images

```
# Prediction results of 20 test set images images:
Predicted starting 10 labels: [3 8 8 0 4 6 3 6 3 9 0 9 5 7 9 8 5 7 8 6]
Actual starting 10 labels:    [3 8 8 0 6 6 1 6 3 1 0 9 5 7 9 8 5 7 8 6]
```

- Now we plot the conclusion matrix and comparison of predicted and actual labels.



OLD ARCHITECTURE		
Layer (type)	**Output Shape**	**Param #**
conv2d (Conv2D)	(None, 30, 30, 32)	896
max_pooling2d (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d (Conv2D)	(None, 13, 13, 64)	18496
max_pooling2d (MaxPooling2D)	(None, 6, 6, 64)	0
flatten (Flatten)	(None, 2304)	0
dense (Dense)	(None, 64)	147520
dense (Dense)	(None, 10)	650

```
Total params: 167,562
Trainable params: 167,562
Non-trainable params: 0
```

NEW ARCHITECTURE		
Layer (type)	**Output Shape**	**Param #**
conv2d (Conv2D)	(None, 32, 32, 16)	432
batch_normalization (BatchNormalization)	(None, 32, 32, 16)	64
max_pooling2d (MaxPooling2D)	(None, 16, 16, 16)	0
conv2d (Conv2D)	(None, 16, 16, 32)	4608
batch_normalization (BatchNormalization)	(None, 16, 16, 32)	128
max_pooling2d (MaxPooling2D)	(None, 8, 8, 32)	0
conv2d (Conv2D)	(None, 8, 8, 64)	18432
batch_normalization (BatchNormalization)	(None, 8, 8, 64)	256
max_pooling2d (MaxPooling2D)	(None, 4, 4, 64)	0
dropout (Dropout)	(None, 4, 4, 64)	0
flatten (Flatten)	(None, 1024)	0
dense (Dense)	(None, 512)	524800
dropout (Dropout)	(None, 512)	0
dense (Dense)	(None, 10)	5130

```
Total params: 553,850
Trainable params: 553,626
Non-trainable params: 224
```

➢ CONCLUSION:

In the process of finishing this assignment, we initially developed an architecture that solely employed convolution and max pooling layers. This approach yielded a test set accuracy of only **68%**. Nonetheless, we improved the accuracy to **78%** by normalizing the data and enhancing the architecture through the inclusion of techniques such as batch normalization, data augmentation, and dropout layers.

THE END.