Ulugbek_st125457_hw4

August 31, 2024

1 Homework 4

1.0.1 Imports

[ ]:
```
from tensorflow.keras.datasets import cifar10
from tensorflow import keras
from tensorflow.keras import layers

from sklearn.metrics import classification_report

import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

1.0.2 Data load

[ ]:
```
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
```

[ ]:
```
X_train.shape, X_test.shape
```

[ ]:
```
((50000, 32, 32, 3), (10000, 32, 32, 3))
```

1.0.3 Normalization

[ ]:
```
X_train = X_train.astype("float") / 255.0
X_test = X_test.astype("float") / 255.0
```

1.0.4 Labeling categorical

[ ]:
```
# One hot encoding
y_train = keras.utils.to_categorical(y_train, 10)
y_test = keras.utils.to_categorical(y_test, 10)
```

1.0.5 Model training

[ ]:
```
model = keras.Sequential()
# CONV => RELU => CONV => RELU => POOL => DROPOUT
model.add(layers.Conv2D(32, (3, 3), padding='same', activation='relu',␣
↪input_shape=X_train.shape[1:]))
model.add(layers.Conv2D(32, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D(pool_size=(2, 2)))
model.add(layers.Dropout(0.1))

# CONV => RELU => CONV => RELU => POOL => DROPOUT
model.add(layers.Conv2D(64, (3, 3), padding='same', activation='relu'))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D(pool_size=(2, 2)))
model.add(layers.Dropout(0.1))

# FLATTERN => DENSE => RELU => DROPOUT
model.add(layers.Flatten())
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dropout(0.2))

# a softmax classifier
model.add(layers.Dense(10, activation='softmax'))
```

```
d:\DataScience\Anaconda3\envs\dl4cv\lib\site-
packages\keras\src\layers\convolutional\base_conv.py:107: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

[ ]:
```
model.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 32, 32, 32)	896
conv2d_5 (Conv2D)	(None, 30, 30, 32)	9,248
max_pooling2d_2 (MaxPooling2D)	(None, 15, 15, 32)	0
dropout_3 (Dropout)	(None, 15, 15, 32)	0
conv2d_6 (Conv2D)	(None, 15, 15, 64)	18,496

```
conv2d_7 (Conv2D)                 (None, 13, 13, 64)           36,928

max_pooling2d_3 (MaxPooling2D)    (None, 6, 6, 64)                  0

dropout_4 (Dropout)               (None, 6, 6, 64)                  0

flatten_1 (Flatten)               (None, 2304)                      0

dense_2 (Dense)                   (None, 512)               1,180,160

dropout_5 (Dropout)               (None, 512)                       0

dense_3 (Dense)                   (None, 10)                    5,130
```

Total params: 1,250,858 (4.77 MB)

Trainable params: 1,250,858 (4.77 MB)

Non-trainable params: 0 (0.00 B)

[ ]:
```
model.compile(optimizer='sgd', loss='categorical_crossentropy',␣
↪metrics=['accuracy'])
```

[ ]:
```
model.fit(X_train, y_train, epochs=35, batch_size=32)
```

```
Epoch 1/35
1563/1563            37s 23ms/step -
accuracy: 0.2146 - loss: 2.1088
Epoch 2/35
1563/1563            35s 23ms/step -
accuracy: 0.4009 - loss: 1.6573
Epoch 3/35
1563/1563            36s 23ms/step -
accuracy: 0.4674 - loss: 1.4752
Epoch 4/35
1563/1563            36s 23ms/step -
accuracy: 0.5184 - loss: 1.3443
Epoch 5/35
1563/1563            37s 24ms/step -
accuracy: 0.5564 - loss: 1.2509
Epoch 6/35
1563/1563            37s 24ms/step -
accuracy: 0.5887 - loss: 1.1661
Epoch 7/35
1563/1563            37s 24ms/step -
```

```
accuracy: 0.6136 - loss: 1.0909
Epoch 8/35
1563/1563            37s 24ms/step -
accuracy: 0.6402 - loss: 1.0134
Epoch 9/35
1563/1563            37s 24ms/step -
accuracy: 0.6648 - loss: 0.9615
Epoch 10/35
1563/1563            37s 24ms/step -
accuracy: 0.6809 - loss: 0.9081
Epoch 11/35
1563/1563            38s 24ms/step -
accuracy: 0.6970 - loss: 0.8596
Epoch 12/35
1563/1563            38s 24ms/step -
accuracy: 0.7153 - loss: 0.8076
Epoch 13/35
1563/1563            37s 24ms/step -
accuracy: 0.7291 - loss: 0.7642
Epoch 14/35
1563/1563            38s 24ms/step -
accuracy: 0.7469 - loss: 0.7261
Epoch 15/35
1563/1563            38s 24ms/step -
accuracy: 0.7650 - loss: 0.6765
Epoch 16/35
1563/1563            37s 24ms/step -
accuracy: 0.7718 - loss: 0.6386
Epoch 17/35
1563/1563            38s 24ms/step -
accuracy: 0.7854 - loss: 0.6051
Epoch 18/35
1563/1563            38s 24ms/step -
accuracy: 0.8018 - loss: 0.5605
Epoch 19/35
1563/1563            38s 24ms/step -
accuracy: 0.8169 - loss: 0.5245
Epoch 20/35
1563/1563            38s 24ms/step -
accuracy: 0.8231 - loss: 0.4939
Epoch 21/35
1563/1563            38s 24ms/step -
accuracy: 0.8372 - loss: 0.4576
Epoch 22/35
1563/1563            38s 24ms/step -
accuracy: 0.8502 - loss: 0.4180
Epoch 23/35
1563/1563            39s 25ms/step -
```

```
accuracy: 0.8567 - loss: 0.4012
Epoch 24/35
1563/1563            37s 24ms/step -
accuracy: 0.8677 - loss: 0.3690
Epoch 25/35
1563/1563            38s 24ms/step -
accuracy: 0.8771 - loss: 0.3440
Epoch 26/35
1563/1563            38s 24ms/step -
accuracy: 0.8864 - loss: 0.3201
Epoch 27/35
1563/1563            37s 24ms/step -
accuracy: 0.8932 - loss: 0.3009
Epoch 28/35
1563/1563            38s 25ms/step -
accuracy: 0.9046 - loss: 0.2704
Epoch 29/35
1563/1563            38s 24ms/step -
accuracy: 0.9089 - loss: 0.2561
Epoch 30/35
1563/1563            38s 25ms/step -
accuracy: 0.9125 - loss: 0.2408
Epoch 31/35
1563/1563            38s 24ms/step -
accuracy: 0.9205 - loss: 0.2238
Epoch 32/35
1563/1563            37s 24ms/step -
accuracy: 0.9240 - loss: 0.2097
Epoch 33/35
1563/1563            38s 24ms/step -
accuracy: 0.9317 - loss: 0.1944
Epoch 34/35
1563/1563            38s 24ms/step -
accuracy: 0.9333 - loss: 0.1839
Epoch 35/35
1563/1563            38s 24ms/step -
accuracy: 0.9423 - loss: 0.1664
```

[ ]:
```
<keras.src.callbacks.history.History at 0x2444d8089d0>
```

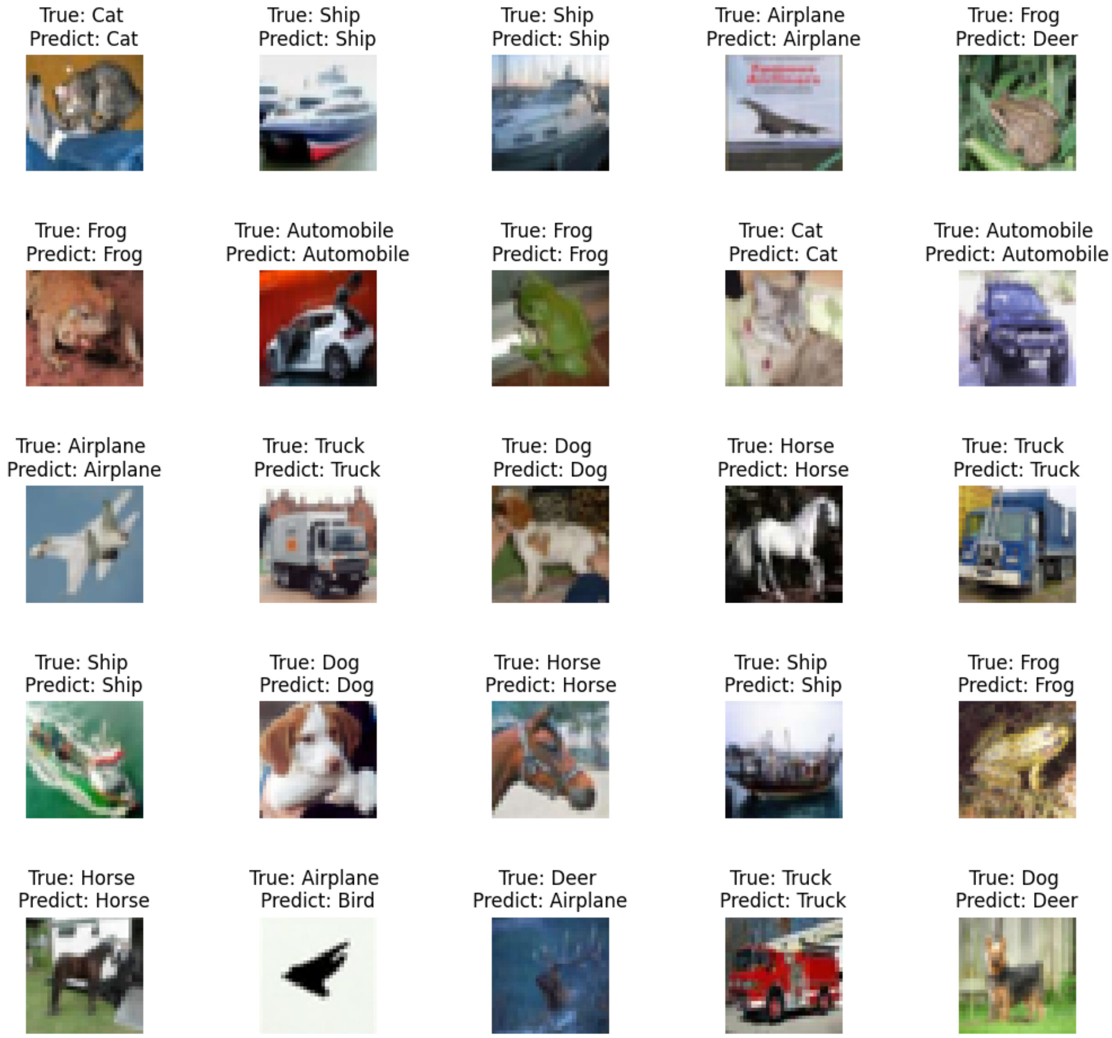
[ ]:
```python
scores = model.evaluate(X_test, y_test, verbose=1)
print('Test loss:', scores[0])
print('Test accuracy:', scores[1])

# make prediction.

pred = model.predict(X_test)
```

```
313/313              3s 8ms/step -
accuracy: 0.7565 - loss: 0.9613
Test loss: 0.9733807444572449
Test accuracy: 0.7541999816894531
313/313              3s 8ms/step
```

[ ]:
```python
Y_pred_classes = np.argmax(pred, axis=1)
# Convert validation observations to one hot vectors
Y_true = np.argmax(y_test, axis=1)

print(classification_report(Y_pred_classes, Y_true))
```

```
              precision    recall  f1-score   support

           0       0.77      0.80      0.78       968
           1       0.85      0.87      0.86       971
           2       0.62      0.74      0.68       841
           3       0.60      0.55      0.57      1098
           4       0.74      0.71      0.73      1046
           5       0.66      0.65      0.66      1008
           6       0.82      0.81      0.81      1014
           7       0.76      0.85      0.80       892
           8       0.86      0.84      0.85      1022
           9       0.87      0.76      0.81      1140

    accuracy                           0.75     10000
   macro avg       0.75      0.76      0.75     10000
weighted avg       0.76      0.75      0.75     10000
```

[ ]:
```python
labels = ['Airplane', 'Automobile', 'Bird', 'Cat', 'Deer', 'Dog', 'Frog',␣
↪'Horse', 'Ship', 'Truck']

R = 5
C = 5
fig, axes = plt.subplots(R, C, figsize=(12,12))
axes = axes.ravel()

for i in np.arange(0, R*C):
    axes[i].imshow(X_test[i])
    axes[i].set_title("True: %s \nPredict: %s" % (labels[Y_true[i]],␣
↪labels[Y_pred_classes[i]]))
    axes[i].axis('off')
    plt.subplots_adjust(wspace=1)
```


True: Cat
Predict: Cat
True: Ship
Predict: Ship
True: Ship
Predict: Ship
True: Airplane
Predict: Airplane
True: Frog
Predict: Deer
True: Frog
Predict: Frog
True: Automobile
Predict: Automobile
True: Frog
Predict: Frog
True: Cat
Predict: Cat
True: Automobile
Predict: Automobile
True: Airplane
Predict: Airplane
True: Truck
Predict: Truck
True: Dog
Predict: Dog
True: Horse
Predict: Horse
True: Truck
Predict: Truck
True: Ship
Predict: Ship
True: Dog
Predict: Dog
True: Horse
Predict: Horse
True: Ship
Predict: Ship
True: Frog
Predict: Frog
True: Horse
Predict: Horse
True: Airplane
Predict: Bird
True: Deer
Predict: Airplane
True: Truck
Predict: Truck
True: Dog
Predict: Deer