# ICP_7 : https://github.com/jxc84430/NeuralNetworks/tree/main/ICP_7

## Spring 2023 : CS5720

Neural Networks & Deep Learning ICP_7 : Jahnavi Chadalavada (700728443)

Deep Learning Image Classification with CNN

1. Training the model
2. Evaluating the model

Programming elements:

1. About CNN
2. Hyperparameters of CNN
3. Image classification with CNN

## 1. Follow the instruction below and then report how the performance changed(apply all at once)

• Convolutional input layer, 32 feature maps with a size of 3×3 and a rectifier activation function. • Dropout layer at 20%.

• Convolutional layer, 32 feature maps with a size of 3×3 and a rectifier activation function.• Max Pool layer with size 2×2.

• Convolutional layer, 64 feature maps with a size of 3×3 and a rectifier activation function.• Dropout layer at 20%.

• Convolutional layer, 64 feature maps with a size of 3×3 and a rectifier activation function.• Max Pool layer with size 2×2.

• Convolutional layer, 128 feature maps with a size of 3×3 and a rectifier activation function.• Dropout layer at 20%.

• Convolutional layer,128 feature maps with a size of 3×3 and a rectifier activation function.• Max Pool layer with size 2×2.

• Flatten layer.• Dropout layer at 20%.

• Fully connected layer with 1024 units and a rectifier activation function.• Dropout layer at 20%.

• Fully connected layer with 512 units and a rectifier activation function.• Dropout layer at 20%.

• Fully connected output layer with 10 units and a Softmax activation function

```
[ ]
```

## Did the performance change?

Model Accuracy has reduced from 90% to 87% where as validation accuracy improved from 69% to 79%.

Execution time Increased.

Model predicted first four test images correctly with these changes where as previous code predicted one image incorrectly.

```
[10] # Simple CNN model for CIFAR-10
     import numpy
     from keras.datasets import cifar10
     from keras.models import Sequential
     from keras.layers import Dense
     from keras.layers import Dropout
     from keras.layers import Flatten
     from keras.constraints import maxnorm
     from keras.optimizers import SGD
```

```
[10] from keras.layers.convolutional import Conv2D
     from keras.layers.convolutional import MaxPooling2D
     from keras.utils import np_utils
     import matplotlib.pyplot as plt

     # fix random seed for reproducibility
     seed = 7
     numpy.random.seed(seed)

     # load data
     (X_train, y_train), (X_test, y_test) = cifar10.load_data()

     # normalize inputs from 0-255 to 0.0-1.0
     X_train = X_train.astype('float32')
     X_test = X_test.astype('float32')
     X_train = X_train / 255.0
     X_test = X_test / 255.0

     # one hot encode outputs
     y_train = np_utils.to_categorical(y_train)
     y_test = np_utils.to_categorical(y_test)
     num_classes = y_test.shape[1]

     # # Create the model
     model = Sequential()
     model.add(Conv2D(32, (3, 3), input_shape=(32, 32,3), padding='same', activation='relu', kernel_constraint=maxnorm(3)))
     model.add(Dropout(0.2))
     model.add(Conv2D(32, (3, 3), activation='relu', padding='same', kernel_constraint=maxnorm(3)))
     model.add(MaxPooling2D(pool_size=(2, 2)))
     model.add(Conv2D(64, (3, 3), input_shape=(32, 32,3), padding='same', activation='relu', kernel_constraint=maxnorm(3)))
```

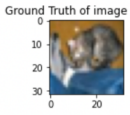
[10]
```
model.add(Dropout(0.2))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same', kernel_constraint=maxnorm(3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, (3, 3), input_shape=(32, 32,3), padding='same', activation='relu', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same', kernel_constraint=maxnorm(3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dropout(0.2))
model.add(Dense(1024, activation='relu', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))

# Compile model
epochs = 25
lrate = 0.01
decay = lrate/epochs
sgd = SGD(lr=lrate, momentum=0.9, decay=decay, nesterov=False)
model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])
print(model.summary())

# Fit the model
history=model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=epochs, batch_size=32)

# Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

[10]
```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_12 (Conv2D)          (None, 32, 32, 32)        896

 dropout_12 (Dropout)        (None, 32, 32, 32)        0

 conv2d_13 (Conv2D)          (None, 32, 32, 32)        9248

 max_pooling2d_6 (MaxPooling  (None, 16, 16, 32)       0
 2D)

 conv2d_14 (Conv2D)          (None, 16, 16, 64)        18496

 dropout_13 (Dropout)        (None, 16, 16, 64)        0

 conv2d_15 (Conv2D)          (None, 16, 16, 64)        36928

 max_pooling2d_7 (MaxPooling  (None, 8, 8, 64)         0
 2D)

 conv2d_16 (Conv2D)          (None, 8, 8, 128)         73856

 dropout_14 (Dropout)        (None, 8, 8, 128)         0

 conv2d_17 (Conv2D)          (None, 8, 8, 128)         147584

 max_pooling2d_8 (MaxPooling  (None, 4, 4, 128)        0
 2D)

 flatten_2 (Flatten)         (None, 2048)              0

[10] dropout_15 (Dropout)        (None, 2048)              0

 dense_6 (Dense)             (None, 1024)              2098176

 dropout_16 (Dropout)        (None, 1024)              0

 dense_7 (Dense)             (None, 512)               524800

 dropout_17 (Dropout)        (None, 512)               0

 dense_8 (Dense)             (None, 10)                5130

=================================================================
Total params: 2,915,114
Trainable params: 2,915,114
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/25
1563/1563 [==============================] - 500s 318ms/step - loss: 1.8548 - accuracy: 0.3132 - val_loss: 1.5085 - val_accuracy: 0.4557
Epoch 2/25
1563/1563 [==============================] - 501s 320ms/step - loss: 1.4169 - accuracy: 0.4830 - val_loss: 1.2830 - val_accuracy: 0.5396
Epoch 3/25
1563/1563 [==============================] - 489s 313ms/step - loss: 1.2100 - accuracy: 0.5645 - val_loss: 1.1022 - val_accuracy: 0.6039
Epoch 4/25
1563/1563 [==============================] - 492s 315ms/step - loss: 1.0610 - accuracy: 0.6237 - val_loss: 0.9752 - val_accuracy: 0.6509
Epoch 5/25
1563/1563 [==============================] - 485s 310ms/step - loss: 0.9478 - accuracy: 0.6630 - val_loss: 0.8548 - val_accuracy: 0.7001
Epoch 6/25
1563/1563 [==============================] - 484s 310ms/step - loss: 0.8567 - accuracy: 0.6956 - val_loss: 0.8513 - val_accuracy: 0.7006
Epoch 7/25
1563/1563 [==============================] - 486s 311ms/step - loss: 0.7906 - accuracy: 0.7197 - val_loss: 0.8105 - val_accuracy: 0.7205
```

```
[10] Epoch 8/25
1563/1563 [==============================] - 471s 301ms/step - loss: 0.7329 - accuracy: 0.7395 - val_loss: 0.7356 - val_accuracy: 0.7418
Epoch 9/25
1563/1563 [==============================] - 466s 298ms/step - loss: 0.6885 - accuracy: 0.7586 - val_loss: 0.7237 - val_accuracy: 0.7477
Epoch 10/25
1563/1563 [==============================] - 477s 305ms/step - loss: 0.6466 - accuracy: 0.7714 - val_loss: 0.7029 - val_accuracy: 0.7531
Epoch 11/25
1563/1563 [==============================] - 483s 309ms/step - loss: 0.6132 - accuracy: 0.7843 - val_loss: 0.6732 - val_accuracy: 0.7604
Epoch 12/25
1563/1563 [==============================] - 469s 300ms/step - loss: 0.5802 - accuracy: 0.7954 - val_loss: 0.6541 - val_accuracy: 0.7717
Epoch 13/25
1563/1563 [==============================] - 482s 308ms/step - loss: 0.5554 - accuracy: 0.8035 - val_loss: 0.6496 - val_accuracy: 0.7752
Epoch 14/25
1563/1563 [==============================] - 477s 305ms/step - loss: 0.5341 - accuracy: 0.8113 - val_loss: 0.6422 - val_accuracy: 0.7785
Epoch 15/25
1563/1563 [==============================] - 476s 305ms/step - loss: 0.5106 - accuracy: 0.8204 - val_loss: 0.6271 - val_accuracy: 0.7851
Epoch 16/25
1563/1563 [==============================] - 478s 306ms/step - loss: 0.4825 - accuracy: 0.8288 - val_loss: 0.6279 - val_accuracy: 0.7858
Epoch 17/25
1563/1563 [==============================] - 476s 305ms/step - loss: 0.4630 - accuracy: 0.8345 - val_loss: 0.6322 - val_accuracy: 0.7812
Epoch 18/25
1563/1563 [==============================] - 464s 297ms/step - loss: 0.4479 - accuracy: 0.8416 - val_loss: 0.6233 - val_accuracy: 0.7857
Epoch 19/25
1563/1563 [==============================] - 462s 296ms/step - loss: 0.4289 - accuracy: 0.8456 - val_loss: 0.6181 - val_accuracy: 0.7906
Epoch 20/25
1563/1563 [==============================] - 462s 296ms/step - loss: 0.4168 - accuracy: 0.8524 - val_loss: 0.6097 - val_accuracy: 0.7922
Epoch 21/25
1563/1563 [==============================] - 478s 306ms/step - loss: 0.4023 - accuracy: 0.8576 - val_loss: 0.6375 - val_accuracy: 0.7855
Epoch 22/25
1563/1563 [==============================] - 476s 305ms/step - loss: 0.3892 - accuracy: 0.8630 - val_loss: 0.6121 - val_accuracy: 0.7949
Epoch 23/25
1563/1563 [==============================] - 476s 304ms/step - loss: 0.3739 - accuracy: 0.8668 - val_loss: 0.6053 - val_accuracy: 0.7941
Epoch 24/25
1563/1563 [==============================] - 481s 308ms/step - loss: 0.3630 - accuracy: 0.8706 - val_loss: 0.6133 - val_accuracy: 0.7960
Epoch 25/25
 750/1563 [=============>................] - ETA: 3:58 - loss: 0.3475 - accuracy: 0.8777
```

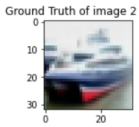
2. Predict the first 4 images of the test data using the above model. Then, compare with the actual label for those 4 images to check whether or not the model has predicted correctly.

```
[12] # plotting first 4 images of the test data and displaying predicted output
     import numpy as np
     import matplotlib.pyplot as plt

     images = ["Airplane","Automobile","Bird","Cat","Deer","Dog","Frog","Horse","Ship","Truck"]
     plt.subplots_adjust(left=0.1,bottom=0.1, right=0.9,top=0.9,wspace=0.8,hspace=0.8)
     for i in range(4):
       yhat=model.predict(X_test[i].reshape(1, 32, 32, 3),verbose=0)
       predicted_output=np.argmax(yhat,axis=1)
       plt.subplot(2,2,i+1)
       plt.imshow(X_test[i].reshape(32, 32,3))
       plt.title('Ground Truth of image {} : {}'.format(i+1,images[np.where(y_test[i] == 1)[0][0]]))
       plt.show()
       print(f"\033[1;31m Ground truth = \033[1;34m {images[np.where(y_test[i] == 1)[0][0]]}    \033[1;31m Model prediction = \033[1;35m {images[predicted_output
```



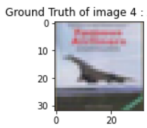

**Ground truth = Cat  Model prediction = Cat**

[12]




**Ground truth = Ship  Model prediction = Ship**


**Ground truth = Ship  Model prediction = Ship**




**Ground truth = Airplane  Model prediction = Airplane**

### 3. Visualize Loss and Accuracy using the history object

```
[13] # plot history for accuracy
     plt.plot(history.history['accuracy'])
     plt.plot(history.history['val_accuracy'])
     plt.title('model accuracy')
     plt.ylabel('accuracy')
     plt.xlabel('epoch')
     plt.legend(['train', 'validation'], loc='lower right')
     plt.show()
     # plot history for loss
     plt.plot(history.history['loss'])
     plt.plot(history.history['val_loss'])
     plt.title('model loss')
     plt.ylabel('loss')
     plt.xlabel('epoch')
     plt.legend(['train', 'validation'], loc='upper right')
     plt.show()
```

[13]