Spring 2024: CS5720

Neural Networks & Deep Learning - ICP-8

NAME:NANDIMANDALAM VENKATA VINAY VARMA

STUDENT ID:700745193

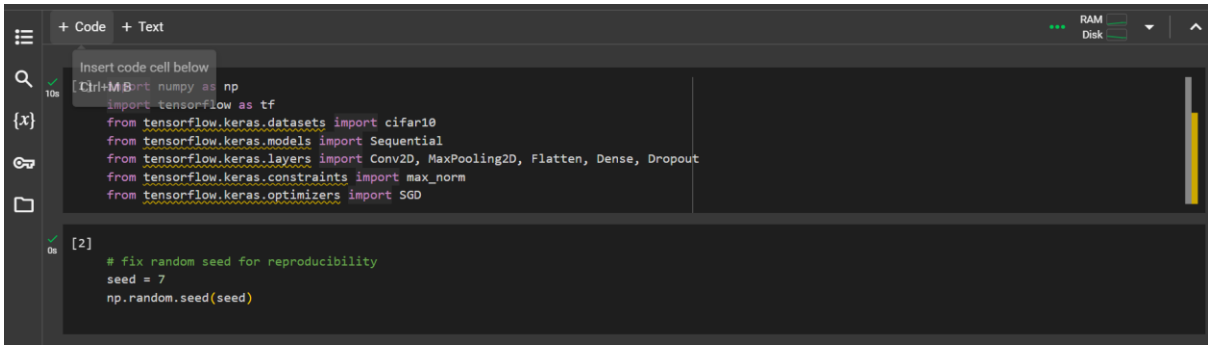
GitHub Link:

https://github.com/venkatavinayvarma/NeuralNetworks_ICP8.git

GitHub Link:

Video Link: https://drive.google.com/drive/folders/1B0X1eq38WGeVXGh2-kyPpdM1e71SFWM5?usp=sharing

Use Case Description: Predicting the diabetes disease Programming elements: Keras Basics In class programming: 1. Use the use case in the class: a. Add more Dense layers to the existing code and check how the accuracy changes. 2. Change the data source to Breast Cancer dataset * available in the source code folder and make required changes. Report accuracy of the model. 3. Normalize the data before feeding the data to the model and check how the normalization change your accuracy (code given below). from sklearn.preprocessing import StandardScaler sc = StandardScaler() Breast Cancer dataset is designated to predict if a patient has Malignant (M) or Benign = B cancer In class programming: Use Image Classification on the hand written digits data set (mnist) 1. Plot the loss and accuracy for both training data and validation data using the history object in the source code. 2. Plot one of the images in the test data, and then do inferencing to check what is the prediction of the model on that single image. 3. We had used 2 hidden layers and Relu activation. Try to change the number of hidden layer and the activation to tanh or sigmoid and see what happens. 4. Run the same code without scaling the images and check the performance?



```
[3]  # 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
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 2s 0us/step
```

```
[4]  # one-hot encode outputs
     #num_classes = 10  # Since CIFAR-10 has 10 classes
     y_train = tf.keras.utils.to_categorical(y_train, num_classes = 10)
     y_test = tf.keras.utils.to_categorical(y_test, num_classes = 10)
     #num_classes = y_test.shape[1]
```

```
     # Create the model
     model = Sequential()
     model.add(Conv2D(32, (3, 3), input_shape=(32, 32, 3), activation='relu', padding='same', kernel_constraint=max_norm(3)))
     model.add(Dropout(0.2))
     model.add(Conv2D(32, (3, 3), activation='relu', padding='same', kernel_constraint=max_norm(3)))
     model.add(MaxPooling2D(pool_size=(2, 2)))
     model.add(Conv2D(64, (3, 3), activation='relu', padding='same', kernel_constraint=max_norm(3)))
     model.add(Dropout(0.2))
     model.add(Conv2D(64, (3, 3), activation='relu', padding='same', kernel_constraint=max_norm(3)))
     model.add(MaxPooling2D(pool_size=(2, 2)))
     model.add(Conv2D(128, (3, 3), activation='relu', padding='same', kernel_constraint=max_norm(3)))
     model.add(Dropout(0.2))
     model.add(Conv2D(128, (3, 3), activation='relu', padding='same', kernel_constraint=max_norm(3)))
     model.add(MaxPooling2D(pool_size=(2, 2)))
     model.add(Flatten())
     model.add(Dropout(0.2))
     model.add(Dense(1024, activation='relu', kernel_constraint=max_norm(3)))
     model.add(Dropout(0.2))
     model.add(Dense(512, activation='relu', kernel_constraint=max_norm(3)))
     model.add(Dropout(0.2))
     model.add(Dense(10, activation='softmax'))
```

```
[6]  # Compile model
     epochs = 5
     lrate = 0.01
     sgd = SGD(lr=lrate, momentum=0.9, nesterov=False)  # Remove decay parameter
     model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])
     print(model.summary())
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.SGD.
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 32)        896

 dropout (Dropout)           (None, 32, 32, 32)        0

 conv2d_1 (Conv2D)           (None, 32, 32, 32)        9248

 max_pooling2d (MaxPooling2  (None, 16, 16, 32)        0
 D)

 conv2d_2 (Conv2D)           (None, 16, 16, 64)        18496

 dropout_1 (Dropout)         (None, 16, 16, 64)        0

 conv2d_3 (Conv2D)           (None, 16, 16, 64)        36928

 max_pooling2d_1 (MaxPoolin  (None, 8, 8, 64)          0
```

```
conv2d_4 (Conv2D)           (None, 8, 8, 128)         73856

dropout_2 (Dropout)         (None, 8, 8, 128)         0

conv2d_5 (Conv2D)           (None, 8, 8, 128)         147584

max_pooling2d_2 (MaxPoolin  (None, 4, 4, 128)         0
g2D)

flatten (Flatten)           (None, 2048)              0

dropout_3 (Dropout)         (None, 2048)              0

dense (Dense)               (None, 1024)              2098176

dropout_4 (Dropout)         (None, 1024)              0

dense_1 (Dense)             (None, 512)               524800

dropout_5 (Dropout)         (None, 512)               0

dense_2 (Dense)             (None, 10)                5130

=================================================================
Total params: 2915114 (11.12 MB)
Trainable params: 2915114 (11.12 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

```
[7] # 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))
```

```
Epoch 1/5
1563/1563 [==============================] - 507s 323ms/step - loss: 1.8599 - accuracy: 0.3126 - val_loss: 1.4817 - val_accuracy: 0.4530
Epoch 2/5
1563/1563 [==============================] - 522s 334ms/step - loss: 1.4269 - accuracy: 0.4810 - val_loss: 1.3270 - val_accuracy: 0.5192
Epoch 3/5
1563/1563 [==============================] - 515s 329ms/step - loss: 1.2139 - accuracy: 0.5643 - val_loss: 1.0599 - val_accuracy: 0.6218
Epoch 4/5
1563/1563 [==============================] - 520s 332ms/step - loss: 1.0560 - accuracy: 0.6258 - val_loss: 1.0832 - val_accuracy: 0.6197
Epoch 5/5
1563/1563 [==============================] - 457s 293ms/step - loss: 0.9435 - accuracy: 0.6672 - val_loss: 0.8670 - val_accuracy: 0.6934
Accuracy: 69.34%
```

```
[8]  # Predict the first 4 test samples
     num_samples_to_predict = 4
     predictions = model.predict(X_test[:num_samples_to_predict])

     # Convert predictions to class labels (assuming one-hot encoding)
     predicted_labels = np.argmax(predictions, axis=1)

     # Convert actual labels to class labels (assuming one-hot encoding)
     actual_labels = np.argmax(y_test[:num_samples_to_predict], axis=1)

     # Print the predicted and actual labels
     print("Predicted Labels:", predicted_labels)
     print("Actual Labels:", actual_labels)
```

```
1/1 [==============================] - 0s 156ms/step
Predicted Labels: [3 1 0 0]
Actual Labels: [3 8 8 0]
```

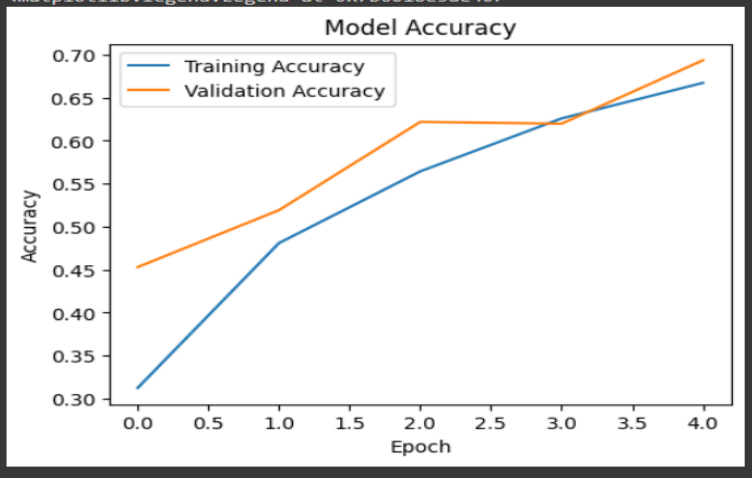
```
[9]  # Compare and print the results
     for i in range(num_samples_to_predict):
         if predicted_labels[i] == actual_labels[i]:
             print(f"Image {i+1}: Predicted Correctly  (Class {predicted_labels[i]})")
         else:
             print(f"Image {i+1}: Predicted Incorrectly  (Predicted Class {predicted_labels[i]}, Actual Class {actual_labels[i]})")
```

```
Image 1: Predicted Correctly  (Class 3)
Image 2: Predicted Incorrectly  (Predicted Class 1, Actual Class 8)
Image 3: Predicted Incorrectly  (Predicted Class 0, Actual Class 8)
Image 4: Predicted Correctly  (Class 0)
```

```python
import matplotlib.pyplot as plt

# Plot training & validation accuracy values
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7b6018e5de40>
```




```python
# Plot training & validation loss values
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.show()
```