dl-lab-experiments-1

December 7, 2023

1 3. Implement a feed forward neural network with three hidden layers for classification on cifar-10 dataset

[ ]:
```
import tensorflow as tf
from keras import models,layers
from keras.datasets import cifar10
from keras.utils import to_categorical
```

[ ]:
```
# Load CIFAR-10 dataset
(train_images, train_labels), (test_images, test_labels) = cifar10.load_data()
```

[ ]:
```
# Normalize pixel values to be between 0 and 1
train_images, test_images = train_images / 255.0, test_images / 255.0
```

[ ]:
```
# Convert labels to one-hot encoding
train_labels = to_categorical(train_labels, 10)
test_labels = to_categorical(test_labels, 10)
```

[ ]:
```
# Define the model
model = models.Sequential()

# Flatten the input for the fully connected layer
model.add(layers.Flatten(input_shape=(32, 32, 3)))

# Three hidden layers with ReLU activation
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(128, activation='relu'))

# Output layer with softmax activation for classification
model.add(layers.Dense(10, activation='softmax'))
```

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

[ ]:
```python
# Display the model summary
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 3072)              0

 dense (Dense)               (None, 512)               1573376

 dense_1 (Dense)             (None, 256)               131328

 dense_2 (Dense)             (None, 128)               32896

 dense_3 (Dense)             (None, 10)                1290

=================================================================
Total params: 1738890 (6.63 MB)
Trainable params: 1738890 (6.63 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[ ]:
```python
# Train the model
history = model.fit(train_images, train_labels, epochs=15,␣
↪validation_data=(test_images, test_labels))
```

```
Epoch 1/15
1563/1563 [==============================] - 8s 5ms/step - loss: 1.8679 -
accuracy: 0.3212 - val_loss: 1.7653 - val_accuracy: 0.3646
Epoch 2/15
1563/1563 [==============================] - 8s 5ms/step - loss: 1.6866 -
accuracy: 0.3927 - val_loss: 1.6459 - val_accuracy: 0.4114
Epoch 3/15
1563/1563 [==============================] - 9s 6ms/step - loss: 1.6033 -
accuracy: 0.4279 - val_loss: 1.5736 - val_accuracy: 0.4340
Epoch 4/15
1563/1563 [==============================] - 7s 5ms/step - loss: 1.5466 -
accuracy: 0.4462 - val_loss: 1.5255 - val_accuracy: 0.4651
Epoch 5/15
1563/1563 [==============================] - 8s 5ms/step - loss: 1.5059 -
accuracy: 0.4618 - val_loss: 1.5525 - val_accuracy: 0.4475
Epoch 6/15
1563/1563 [==============================] - 7s 5ms/step - loss: 1.4727 -
accuracy: 0.4709 - val_loss: 1.5078 - val_accuracy: 0.4558
Epoch 7/15
1563/1563 [==============================] - 8s 5ms/step - loss: 1.4399 -
accuracy: 0.4819 - val_loss: 1.4586 - val_accuracy: 0.4799
```

```
Epoch 8/15
1563/1563 [==============================] - 7s 4ms/step - loss: 1.4140 -
accuracy: 0.4934 - val_loss: 1.4339 - val_accuracy: 0.4904
Epoch 9/15
1563/1563 [==============================] - 7s 5ms/step - loss: 1.3868 -
accuracy: 0.5019 - val_loss: 1.4302 - val_accuracy: 0.4941
Epoch 10/15
1563/1563 [==============================] - 7s 5ms/step - loss: 1.3679 -
accuracy: 0.5093 - val_loss: 1.5061 - val_accuracy: 0.4642
Epoch 11/15
1563/1563 [==============================] - 8s 5ms/step - loss: 1.3364 -
accuracy: 0.5212 - val_loss: 1.4431 - val_accuracy: 0.4831
Epoch 12/15
1563/1563 [==============================] - 8s 5ms/step - loss: 1.3235 -
accuracy: 0.5239 - val_loss: 1.4501 - val_accuracy: 0.4901
Epoch 13/15
1563/1563 [==============================] - 7s 4ms/step - loss: 1.3042 -
accuracy: 0.5305 - val_loss: 1.4099 - val_accuracy: 0.5033
Epoch 14/15
1563/1563 [==============================] - 8s 5ms/step - loss: 1.2855 -
accuracy: 0.5385 - val_loss: 1.4485 - val_accuracy: 0.4904
Epoch 15/15
1563/1563 [==============================] - 7s 4ms/step - loss: 1.2690 -
accuracy: 0.5425 - val_loss: 1.4293 - val_accuracy: 0.4974
```

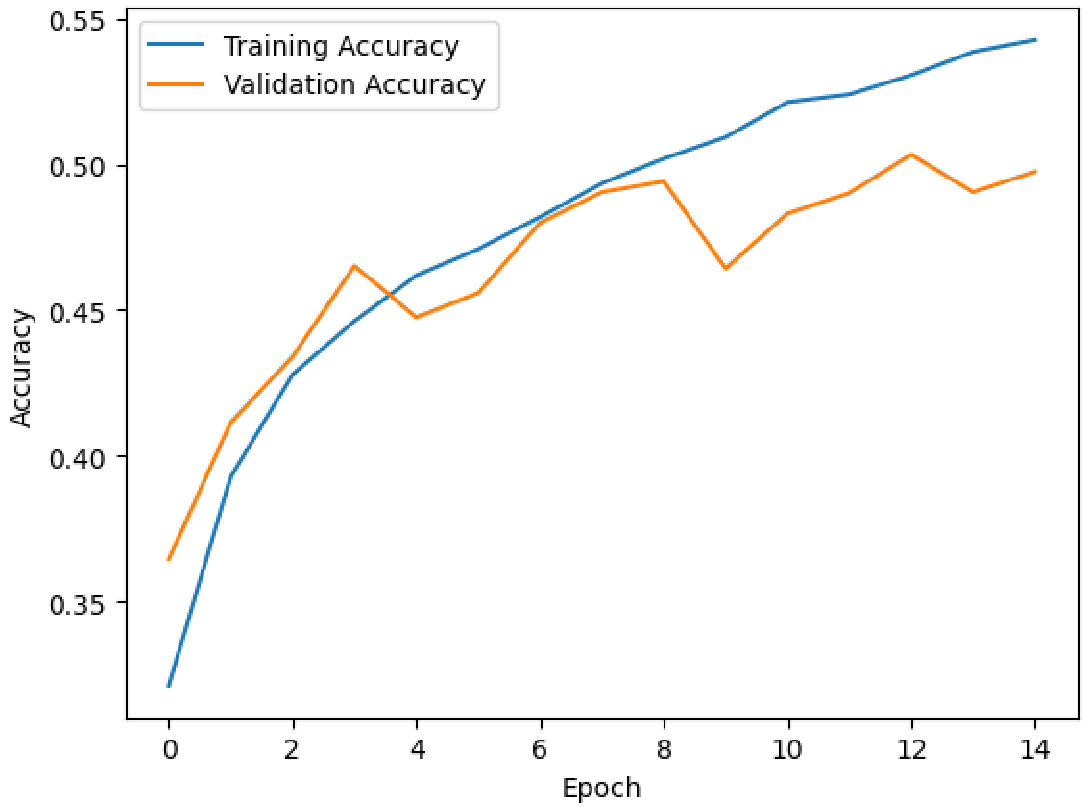
[ ]:

```python
# Evaluate the model
score = model.evaluate(test_images, test_labels)
```

```
313/313 [==============================] - 1s 3ms/step - loss: 1.4293 -
accuracy: 0.4974
```

[ ]:

```python
import matplotlib.pyplot as plt
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



2 4. Analyzing the impact of optimization and weight initialization techniques on neural networks

[ ]:
```
import tensorflow as tf
import numpy as np
from keras import models,layers,optimizers
from keras.datasets import cifar10
from keras.utils import to_categorical
```

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

[ ]:
```
X_train = X_train.astype('float32')/ 255.0
X_test = X_test.astype('float32')/255.0
```

[ ]:
```
X_train.shape
```

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

[ ]:
```
y_train = to_categorical(y_train,10)
y_test = to_categorical(y_test,10)
```

[ ]:
```python
#Xavier Initialization
model1 = models.Sequential()

model1.add(layers.Flatten(input_shape=(32,32,3)))

model1.add(layers.
 ↪Dense(256,activation='relu',kernel_initializer='glorot_uniform'))
model1.add(layers.
 ↪Dense(256,activation='relu',kernel_initializer='glorot_uniform'))

model1.add(layers.
 ↪Dense(10,activation='softmax',kernel_initializer='glorot_uniform'))
```

[ ]:
```python
#Kaiming Initialization
model2 = models.Sequential()

model2.add(layers.Flatten(input_shape=(32,32,3)))

model2.add(layers.Dense(256,activation='relu',kernel_initializer='he_normal'))
model2.add(layers.Dense(128,activation='relu',kernel_initializer='he_normal'))

model2.add(layers.Dense(10,activation='softmax',kernel_initializer='he_normal'))
```

[ ]:
```python
#With dropout Layer
model3 = models.Sequential()
model3.add(layers.Flatten(input_shape=(32,32,3)))

model3.add(layers.
 ↪Dense(256,activation='relu',kernel_initializer='glorot_uniform'))
model3.add(layers.Dropout(0.25))
model3.add(layers.Dense(128,activation='relu'))

model3.add(layers.Dense(10,activation='softmax')
```

[ ]:
```python
# with batch normalization
model4 = models.Sequential()
model4.add(layers.Flatten(input_shape=(32,32,3)))
model4.add(layers.Dense(256,activation='relu'))
model4.add(layers.BatchNormalization())
model4.add(layers.Activation('relu'))
model4.add(layers.Dense(10,activation='softmax'))
```

[ ]:
```python
sgd_optimizer = optimizers.SGD(learning_rate=0.01, momentum=0.9)
model1.compile(optimizer=sgd_optimizer,
              loss='categorical_crossentropy',
              metrics=['accuracy'])
print(model1.summary())
```

```
history1  = model1.
 ↪fit(X_train,y_train,epochs=15,batch_size=32,validation_split=0.2)
score1 = model1.evaluate(X_test,y_test,batch_size=32)
print(score1)
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_5 (Flatten)         (None, 3072)              0

 dense_20 (Dense)            (None, 256)               786688

 dense_21 (Dense)            (None, 256)               65792

 dense_22 (Dense)            (None, 10)                2570

=================================================================
Total params: 855050 (3.26 MB)
Trainable params: 855050 (3.26 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
Epoch 1/15
1250/1250 [==============================] - 5s 4ms/step - loss: 1.2011 -
accuracy: 0.5697 - val_loss: 1.5958 - val_accuracy: 0.4554
Epoch 2/15
1250/1250 [==============================] - 5s 4ms/step - loss: 1.1992 -
accuracy: 0.5675 - val_loss: 1.6043 - val_accuracy: 0.4731
Epoch 3/15
1250/1250 [==============================] - 8s 6ms/step - loss: 1.1797 -
accuracy: 0.5774 - val_loss: 1.6420 - val_accuracy: 0.4659
Epoch 4/15
1250/1250 [==============================] - 11s 8ms/step - loss: 1.1725 -
accuracy: 0.5795 - val_loss: 1.5975 - val_accuracy: 0.4805
Epoch 5/15
1250/1250 [==============================] - 6s 5ms/step - loss: 1.1643 -
accuracy: 0.5820 - val_loss: 1.6569 - val_accuracy: 0.4851
Epoch 6/15
1250/1250 [==============================] - 6s 5ms/step - loss: 1.1522 -
accuracy: 0.5883 - val_loss: 1.6383 - val_accuracy: 0.4823
Epoch 7/15
1250/1250 [==============================] - 5s 4ms/step - loss: 1.1342 -
accuracy: 0.5931 - val_loss: 1.6576 - val_accuracy: 0.4696
Epoch 8/15
1250/1250 [==============================] - 5s 4ms/step - loss: 1.1329 -
accuracy: 0.5934 - val_loss: 1.6258 - val_accuracy: 0.4714
```

```
Epoch 9/15
1250/1250 [==============================] - 6s 5ms/step - loss: 1.1284 -
accuracy: 0.5943 - val_loss: 1.6065 - val_accuracy: 0.4804
Epoch 10/15
1250/1250 [==============================] - 7s 6ms/step - loss: 1.1280 -
accuracy: 0.5945 - val_loss: 1.6600 - val_accuracy: 0.4794
Epoch 11/15
1250/1250 [==============================] - 11s 9ms/step - loss: 1.0999 -
accuracy: 0.6050 - val_loss: 1.6530 - val_accuracy: 0.4806
Epoch 12/15
1250/1250 [==============================] - 5s 4ms/step - loss: 1.0996 -
accuracy: 0.6064 - val_loss: 1.6461 - val_accuracy: 0.4627
Epoch 13/15
1250/1250 [==============================] - 6s 5ms/step - loss: 1.0901 -
accuracy: 0.6094 - val_loss: 1.6737 - val_accuracy: 0.4865
Epoch 14/15
1250/1250 [==============================] - 5s 4ms/step - loss: 1.0911 -
accuracy: 0.6061 - val_loss: 1.6170 - val_accuracy: 0.4868
Epoch 15/15
1250/1250 [==============================] - 5s 4ms/step - loss: 1.0834 -
accuracy: 0.6080 - val_loss: 1.6006 - val_accuracy: 0.4867
313/313 [==============================] - 1s 3ms/step - loss: 1.5901 -
accuracy: 0.4834
[1.5901356935501099, 0.48339998722076416]
```

[ ]:
```python
sgd_optimizer = optimizers.SGD(learning_rate=0.01, momentum=0.9)
model2.compile(optimizer=sgd_optimizer,
              loss='categorical_crossentropy',
              metrics=['accuracy'])
print(model2.summary())
history2  = model2.
 ↪fit(X_train,y_train,epochs=15,batch_size=32,validation_split=0.2)
score2 = model2.evaluate(X_test,y_test,batch_size=128)
print(score2)
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_7 (Flatten)         (None, 3072)              0

 dense_23 (Dense)            (None, 256)               786688

 dense_24 (Dense)            (None, 128)               32896

 dense_25 (Dense)            (None, 10)                1290

=================================================================
```

```
Total params: 820874 (3.13 MB)
Trainable params: 820874 (3.13 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
Epoch 1/15
1250/1250 [==============================] - 6s 4ms/step - loss: 1.8980 -
accuracy: 0.3108 - val_loss: 1.8428 - val_accuracy: 0.3467
Epoch 2/15
1250/1250 [==============================] - 5s 4ms/step - loss: 1.7475 -
accuracy: 0.3696 - val_loss: 1.7079 - val_accuracy: 0.3949
Epoch 3/15
1250/1250 [==============================] - 6s 5ms/step - loss: 1.6806 -
accuracy: 0.3970 - val_loss: 1.6845 - val_accuracy: 0.4003
Epoch 4/15
1250/1250 [==============================] - 5s 4ms/step - loss: 1.6408 -
accuracy: 0.4121 - val_loss: 1.6433 - val_accuracy: 0.4137
Epoch 5/15
1250/1250 [==============================] - 6s 5ms/step - loss: 1.6106 -
accuracy: 0.4227 - val_loss: 1.6555 - val_accuracy: 0.4011
Epoch 6/15
1250/1250 [==============================] - 5s 4ms/step - loss: 1.5970 -
accuracy: 0.4271 - val_loss: 1.6374 - val_accuracy: 0.4168
Epoch 7/15
1250/1250 [==============================] - 5s 4ms/step - loss: 1.5810 -
accuracy: 0.4349 - val_loss: 1.6392 - val_accuracy: 0.4246
Epoch 8/15
1250/1250 [==============================] - 6s 5ms/step - loss: 1.5544 -
accuracy: 0.4430 - val_loss: 1.6078 - val_accuracy: 0.4346
Epoch 9/15
1250/1250 [==============================] - 5s 4ms/step - loss: 1.5450 -
accuracy: 0.4478 - val_loss: 1.6112 - val_accuracy: 0.4334
Epoch 10/15
1250/1250 [==============================] - 6s 5ms/step - loss: 1.5314 -
accuracy: 0.4518 - val_loss: 1.6114 - val_accuracy: 0.4192
Epoch 11/15
1250/1250 [==============================] - 5s 4ms/step - loss: 1.5178 -
accuracy: 0.4579 - val_loss: 1.6373 - val_accuracy: 0.4232
Epoch 12/15
1250/1250 [==============================] - 5s 4ms/step - loss: 1.5006 -
accuracy: 0.4611 - val_loss: 1.6064 - val_accuracy: 0.4356
Epoch 13/15
1250/1250 [==============================] - 6s 5ms/step - loss: 1.4885 -
accuracy: 0.4659 - val_loss: 1.5687 - val_accuracy: 0.4506
Epoch 14/15
1250/1250 [==============================] - 5s 4ms/step - loss: 1.4713 -
accuracy: 0.4740 - val_loss: 1.5912 - val_accuracy: 0.4385
Epoch 15/15
```

```
1250/1250 [==============================] - 5s 4ms/step - loss: 1.4612 -
accuracy: 0.4802 - val_loss: 1.5714 - val_accuracy: 0.4573
79/79 [==============================] - 0s 3ms/step - loss: 1.5333 - accuracy:
0.4648
[1.5333362817764282, 0.46480000019073486]
```

[ ]:
```
sgd_optimizer = optimizers.SGD(learning_rate=0.01, momentum=0.9)
model3.compile(optimizer=sgd_optimizer,
              loss='categorical_crossentropy',
              metrics=['accuracy'])
print(model3.summary())
history3  = model3.
 ↪fit(X_train,y_train,epochs=15,batch_size=32,validation_split=0.2)
score3 = model3.evaluate(X_test,y_test,batch_size=128)
print(score3)
```

```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_9 (Flatten)         (None, 3072)              0

 dense_27 (Dense)            (None, 256)               786688

 dropout_1 (Dropout)         (None, 256)               0

 dense_28 (Dense)            (None, 128)               32896

 dense_29 (Dense)            (None, 10)                1290

 dense_31 (Dense)            (None, 10)                110

=================================================================
Total params: 820984 (3.13 MB)
Trainable params: 820984 (3.13 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
Epoch 1/15
1250/1250 [==============================] - 8s 5ms/step - loss: 2.1377 -
accuracy: 0.1784 - val_loss: 2.0517 - val_accuracy: 0.2064
Epoch 2/15
1250/1250 [==============================] - 6s 5ms/step - loss: 2.0110 -
accuracy: 0.2422 - val_loss: 1.9571 - val_accuracy: 0.2505
Epoch 3/15
1250/1250 [==============================] - 5s 4ms/step - loss: 1.9457 -
accuracy: 0.2711 - val_loss: 1.9030 - val_accuracy: 0.2968
Epoch 4/15
```

```
1250/1250 [==============================] - 6s 5ms/step - loss: 1.8911 -
accuracy: 0.2982 - val_loss: 1.8843 - val_accuracy: 0.3035
Epoch 5/15
1250/1250 [==============================] - 5s 4ms/step - loss: 1.8400 -
accuracy: 0.3206 - val_loss: 1.8207 - val_accuracy: 0.3144
Epoch 6/15
1250/1250 [==============================] - 6s 4ms/step - loss: 1.8117 -
accuracy: 0.3339 - val_loss: 1.7592 - val_accuracy: 0.3613
Epoch 7/15
1250/1250 [==============================] - 6s 5ms/step - loss: 1.7805 -
accuracy: 0.3477 - val_loss: 1.7352 - val_accuracy: 0.3661
Epoch 8/15
1250/1250 [==============================] - 5s 4ms/step - loss: 1.7502 -
accuracy: 0.3584 - val_loss: 1.7276 - val_accuracy: 0.3777
Epoch 9/15
1250/1250 [==============================] - 6s 5ms/step - loss: 1.7311 -
accuracy: 0.3685 - val_loss: 1.7064 - val_accuracy: 0.3892
Epoch 10/15
1250/1250 [==============================] - 6s 4ms/step - loss: 1.7192 -
accuracy: 0.3742 - val_loss: 1.6959 - val_accuracy: 0.3978
Epoch 11/15
1250/1250 [==============================] - 6s 5ms/step - loss: 1.6993 -
accuracy: 0.3808 - val_loss: 1.7066 - val_accuracy: 0.3839
Epoch 12/15
1250/1250 [==============================] - 5s 4ms/step - loss: 1.6832 -
accuracy: 0.3891 - val_loss: 1.6785 - val_accuracy: 0.4030
Epoch 13/15
1250/1250 [==============================] - 5s 4ms/step - loss: 1.6694 -
accuracy: 0.3991 - val_loss: 1.6525 - val_accuracy: 0.4129
Epoch 14/15
1250/1250 [==============================] - 7s 5ms/step - loss: 1.6570 -
accuracy: 0.4005 - val_loss: 1.6324 - val_accuracy: 0.4188
Epoch 15/15
1250/1250 [==============================] - 6s 4ms/step - loss: 1.6527 -
accuracy: 0.3997 - val_loss: 1.6629 - val_accuracy: 0.4046
79/79 [==============================] - 0s 3ms/step - loss: 1.6370 - accuracy:
0.4105
[1.636962652206421, 0.4104999899864197]
```

[ ]:
```
X_train.shape
```

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

[ ]:
```
sgd_optimizer = optimizers.SGD(learning_rate=0.01, momentum=0.9)
model4.compile(optimizer=sgd_optimizer,
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

```python
print(model4.summary())
history4  = model4.
 ↪fit(X_train,y_train,epochs=15,batch_size=128,validation_split=0.2)
score4 = model4.evaluate(X_test,y_test,batch_size=128)
print(score4)
```

```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_11 (Flatten)        (None, 3072)              0

 dense_32 (Dense)            (None, 256)               786688

 batch_normalization_1 (Bat  (None, 256)               1024
 chNormalization)

 activation_1 (Activation)   (None, 256)               0

 dense_33 (Dense)            (None, 10)                2570

=================================================================
Total params: 790282 (3.01 MB)
Trainable params: 789770 (3.01 MB)
Non-trainable params: 512 (2.00 KB)
_________________________________________________________________
None
Epoch 1/15
313/313 [==============================] - 4s 7ms/step - loss: 1.7398 -
accuracy: 0.3937 - val_loss: 1.8273 - val_accuracy: 0.3633
Epoch 2/15
313/313 [==============================] - 2s 5ms/step - loss: 1.5554 -
accuracy: 0.4593 - val_loss: 1.8369 - val_accuracy: 0.3696
Epoch 3/15
313/313 [==============================] - 2s 5ms/step - loss: 1.4874 -
accuracy: 0.4831 - val_loss: 1.6273 - val_accuracy: 0.4134
Epoch 4/15
313/313 [==============================] - 2s 5ms/step - loss: 1.4521 -
accuracy: 0.4927 - val_loss: 1.6635 - val_accuracy: 0.4184
Epoch 5/15
313/313 [==============================] - 2s 6ms/step - loss: 1.4175 -
accuracy: 0.5048 - val_loss: 1.6965 - val_accuracy: 0.4008
Epoch 6/15
313/313 [==============================] - 2s 7ms/step - loss: 1.3947 -
accuracy: 0.5125 - val_loss: 1.8659 - val_accuracy: 0.3850
Epoch 7/15
313/313 [==============================] - 2s 5ms/step - loss: 1.3641 -
```

```
accuracy: 0.5233 - val_loss: 1.7673 - val_accuracy: 0.3930
Epoch 8/15
313/313 [==============================] - 2s 5ms/step - loss: 1.3347 -
accuracy: 0.5315 - val_loss: 1.7314 - val_accuracy: 0.4113
Epoch 9/15
313/313 [==============================] - 2s 5ms/step - loss: 1.3013 -
accuracy: 0.5443 - val_loss: 1.6492 - val_accuracy: 0.4335
Epoch 10/15
313/313 [==============================] - 2s 5ms/step - loss: 1.2766 -
accuracy: 0.5531 - val_loss: 1.7191 - val_accuracy: 0.4187
Epoch 11/15
313/313 [==============================] - 2s 5ms/step - loss: 1.2561 -
accuracy: 0.5607 - val_loss: 1.7300 - val_accuracy: 0.4207
Epoch 12/15
313/313 [==============================] - 2s 6ms/step - loss: 1.2457 -
accuracy: 0.5631 - val_loss: 1.9348 - val_accuracy: 0.4021
Epoch 13/15
313/313 [==============================] - 2s 6ms/step - loss: 1.2302 -
accuracy: 0.5686 - val_loss: 1.8605 - val_accuracy: 0.4095
Epoch 14/15
313/313 [==============================] - 2s 6ms/step - loss: 1.2168 -
accuracy: 0.5719 - val_loss: 1.6793 - val_accuracy: 0.4348
Epoch 15/15
313/313 [==============================] - 2s 5ms/step - loss: 1.1958 -
accuracy: 0.5775 - val_loss: 1.8075 - val_accuracy: 0.4141
79/79 [==============================] - 0s 3ms/step - loss: 1.7979 - accuracy:
0.4188
[1.7978681325912476, 0.4187999963760376]
```

3 5. Digit Classification using CNN Architecture for MNIST Dataset

[18]:
```python
import tensorflow as tf
import numpy as np
from keras import layers, models
from keras.datasets import mnist
from keras.utils import to_categorical
```

[19]:
```python
(X_train,y_train),(X_test,y_test) = mnist.load_data()
```

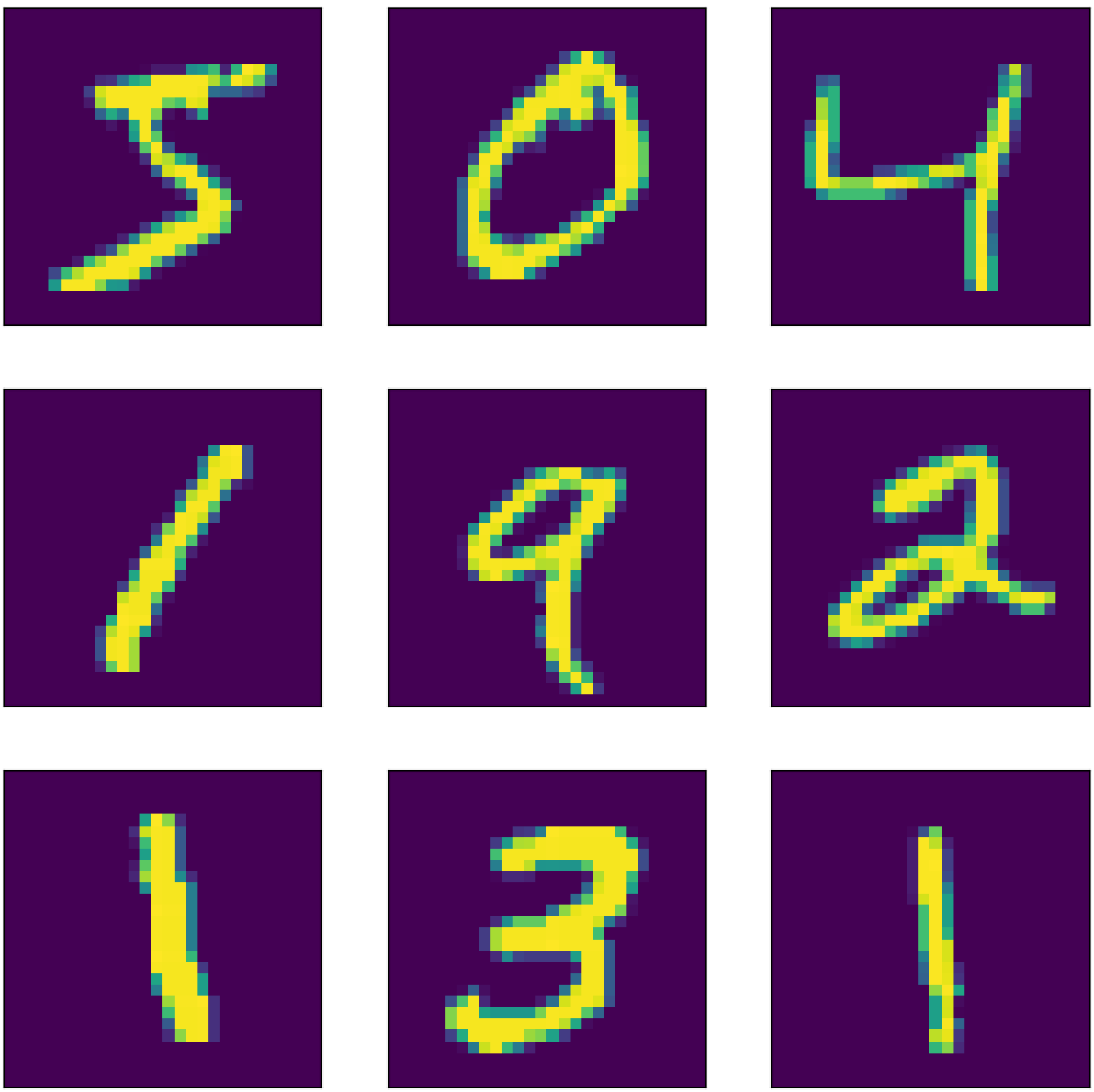
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
```

[20]:
```python
print(X_train.shape)
print(y_train.shape)
```

```
(60000, 28, 28)
(60000,)
```

[29]:
```python
import matplotlib.pyplot as plt

plt.figure(figsize=(10,10))
for i in range(9):
    plt.subplot(3,3,i+1)
    plt.imshow(X_train[i])
    plt.xticks([])
    plt.yticks([])
```



[21]:
```
X_train =  X_train.reshape((60000,28,28,1)).astype('float32')/255.0
 X_test = X_test.reshape((10000,28,28,1)).astype('float32')/255.0

 y_train = to_categorical(y_train)
 y_test = to_categorical(y_test)
```

[24]:
```
# Build the CNN model
model = models.Sequential([
    layers.Conv2D(32,(3,3),activation='relu',input_shape=(28,28,1)),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64,(3,3),activation='relu'),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64,(3,3),activation='relu'),
    layers.Flatten(),
    layers.Dense(64,activation='relu'),
    layers.Dense(10,activation='softmax')
])
model.compile(optimizer='adam',
             loss='categorical_crossentropy',
             metrics=['accuracy'])
print(model.summary())

history = model.fit(X_train,y_train,epochs=15,batch_size=64,validation_split=0.
  ↪2)
```

```
Model: "sequential_6"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_6 (Conv2D)           (None, 26, 26, 32)        320

 max_pooling2d_4 (MaxPoolin  (None, 13, 13, 32)        0
 g2D)

 conv2d_7 (Conv2D)           (None, 11, 11, 64)        18496

 max_pooling2d_5 (MaxPoolin  (None, 5, 5, 64)          0
 g2D)

 conv2d_8 (Conv2D)           (None, 3, 3, 64)          36928

 flatten_2 (Flatten)         (None, 576)               0

 dense_8 (Dense)             (None, 64)                36928

 dense_9 (Dense)             (None, 10)                650
```

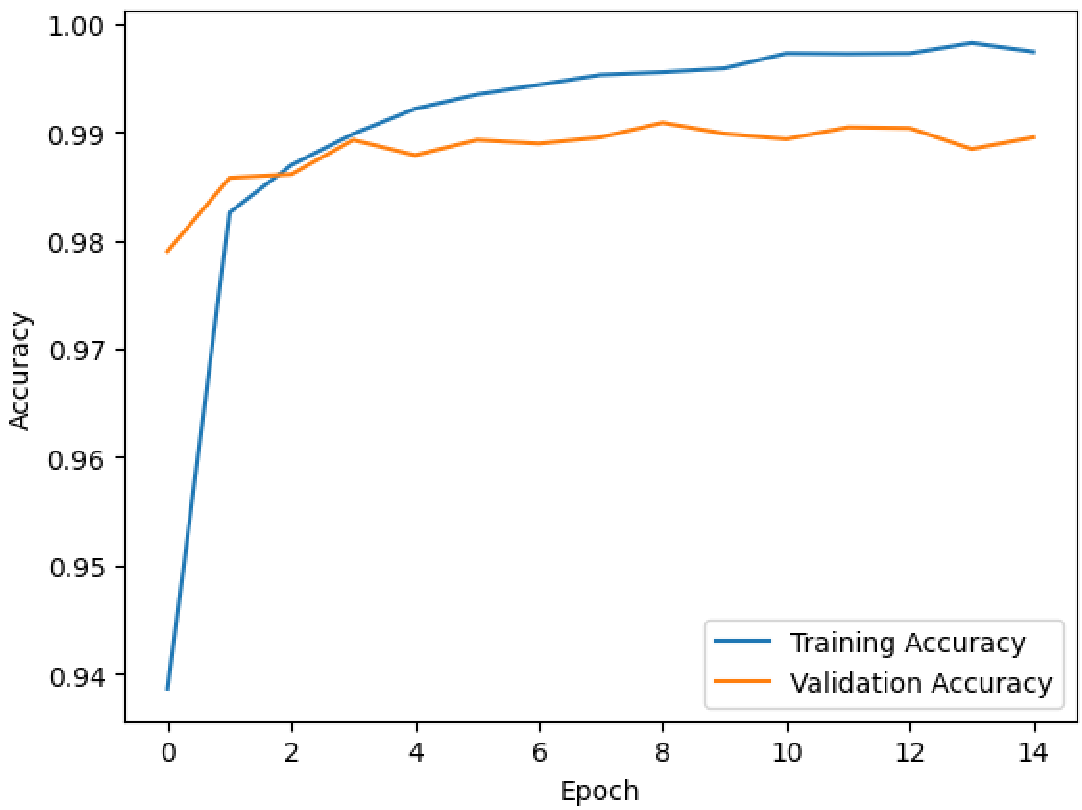
```
=================================================================
Total params: 93322 (364.54 KB)
Trainable params: 93322 (364.54 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
Epoch 1/15
750/750 [==============================] - 56s 74ms/step - loss: 0.1989 -
accuracy: 0.9386 - val_loss: 0.0723 - val_accuracy: 0.9790
Epoch 2/15
750/750 [==============================] - 48s 64ms/step - loss: 0.0550 -
accuracy: 0.9826 - val_loss: 0.0506 - val_accuracy: 0.9858
Epoch 3/15
750/750 [==============================] - 48s 64ms/step - loss: 0.0404 -
accuracy: 0.9869 - val_loss: 0.0449 - val_accuracy: 0.9861
Epoch 4/15
750/750 [==============================] - 50s 66ms/step - loss: 0.0308 -
accuracy: 0.9898 - val_loss: 0.0378 - val_accuracy: 0.9893
Epoch 5/15
750/750 [==============================] - 50s 66ms/step - loss: 0.0241 -
accuracy: 0.9921 - val_loss: 0.0448 - val_accuracy: 0.9878
Epoch 6/15
750/750 [==============================] - 48s 64ms/step - loss: 0.0203 -
accuracy: 0.9934 - val_loss: 0.0384 - val_accuracy: 0.9893
Epoch 7/15
750/750 [==============================] - 51s 68ms/step - loss: 0.0162 -
accuracy: 0.9943 - val_loss: 0.0382 - val_accuracy: 0.9889
Epoch 8/15
750/750 [==============================] - 52s 70ms/step - loss: 0.0135 -
accuracy: 0.9952 - val_loss: 0.0371 - val_accuracy: 0.9895
Epoch 9/15
750/750 [==============================] - 50s 66ms/step - loss: 0.0127 -
accuracy: 0.9955 - val_loss: 0.0368 - val_accuracy: 0.9908
Epoch 10/15
750/750 [==============================] - 50s 67ms/step - loss: 0.0115 -
accuracy: 0.9959 - val_loss: 0.0368 - val_accuracy: 0.9898
Epoch 11/15
750/750 [==============================] - 47s 63ms/step - loss: 0.0081 -
accuracy: 0.9972 - val_loss: 0.0471 - val_accuracy: 0.9893
Epoch 12/15
750/750 [==============================] - 51s 68ms/step - loss: 0.0079 -
accuracy: 0.9972 - val_loss: 0.0411 - val_accuracy: 0.9904
Epoch 13/15
750/750 [==============================] - 50s 66ms/step - loss: 0.0089 -
accuracy: 0.9972 - val_loss: 0.0378 - val_accuracy: 0.9903
Epoch 14/15
750/750 [==============================] - 49s 66ms/step - loss: 0.0059 -
accuracy: 0.9982 - val_loss: 0.0475 - val_accuracy: 0.9884
```

```
Epoch 15/15
750/750 [==============================] - 47s 63ms/step - loss: 0.0071 -
accuracy: 0.9974 - val_loss: 0.0478 - val_accuracy: 0.9895
```

[25]:
```python
# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(X_test,y_test)
print(f'Test accuracy: {test_acc}')
```

```
313/313 [==============================] - 4s 13ms/step - loss: 0.0377 -
accuracy: 0.9906
Test accuracy: 0.9905999898910522
```

[28]:
```python
import matplotlib.pyplot as plt
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend();
```



4 6. Digit classification using pre-trained networks like VGGnet-19 for MNIST dataset and analyse and visualize performance improvements.

```
[ ]: import tensorflow as tf
     import numpy as np
     from keras.datasets import mnist
     from tensorflow.keras.utils import to_categorical
```

```
[ ]: (X_train, y_train), (X_test, y_test) = mnist.load_data()
     print(f'X_train shape: {X_train.shape}')
```

```
X_train shape: (60000, 28, 28)
```

5 7. Implement a simple RNN for review classification using IMDB dataset.

```
[1]: from keras.datasets import imdb
     import tensorflow as tf
     from keras import layers,models,Sequential

     from keras.preprocessing import sequence
     from keras.utils import pad_sequences
```

```
[2]: max_features = 5000
     max_words=500
     (X_train,y_train), (X_test,y_test) = imdb.load_data(maxlen=max_features)
     print(f'{len(X_train)} train sequences\n{len(X_test)} test sequences')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/imdb.npz
17464789/17464789 [==============================] - 1s 0us/step
25000 train sequences
25000 test sequences
```

```
[4]: # pad sequences to fixed length
     X_train = sequence.pad_sequences(X_train,maxlen=max_words)
     X_test = sequence.pad_sequences(X_test,maxlen=max_words)
     print('train data shape: ',X_train.shape)
     print('test data shape: ',X_test.shape)
```

```
train data shape:  (25000, 500)
test data shape:  (25000, 500)
```

```
[5]: model = models.Sequential()
     model.add(layers.Embedding(max_features,32,input_length=max_words))
```

```
model.add(layers.SimpleRNN(100))
model.add(layers.Dense(1,activation='sigmoid'))
model.summary()
```

```
Model: "sequential"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 500, 32)           160000

 simple_rnn (SimpleRNN)      (None, 100)               13300

 dense (Dense)               (None, 1)                 101

=================================================================
Total params: 173401 (677.35 KB)
Trainable params: 173401 (677.35 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[6]:
```
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy']

)
```

[7]:
```
history = model.fit(X_train,y_train,epochs=15,batch_size=64,validation_split=0.
 ↪2)
```

```
Epoch 1/15
313/313 [==============================] - 230s 708ms/step - loss: 0.6843 -
accuracy: 0.5573 - val_loss: 0.6633 - val_accuracy: 0.6072
Epoch 2/15
313/313 [==============================] - 202s 643ms/step - loss: 0.6137 -
accuracy: 0.6628 - val_loss: 0.5813 - val_accuracy: 0.6786
Epoch 3/15
313/313 [==============================] - 192s 616ms/step - loss: 0.5331 -
accuracy: 0.7311 - val_loss: 0.6571 - val_accuracy: 0.6026
Epoch 4/15
313/313 [==============================] - 183s 585ms/step - loss: 0.5762 -
accuracy: 0.7063 - val_loss: 0.6230 - val_accuracy: 0.6370
Epoch 5/15
313/313 [==============================] - 186s 594ms/step - loss: 0.5012 -
accuracy: 0.7538 - val_loss: 0.5484 - val_accuracy: 0.7418
Epoch 6/15
313/313 [==============================] - 180s 576ms/step - loss: 0.4430 -
accuracy: 0.7979 - val_loss: 0.5692 - val_accuracy: 0.7314
```

```
Epoch 7/15
313/313 [==============================] - 182s 582ms/step - loss: 0.4193 -
accuracy: 0.8134 - val_loss: 0.5856 - val_accuracy: 0.7028
Epoch 8/15
313/313 [==============================] - 176s 562ms/step - loss: 0.3712 -
accuracy: 0.8468 - val_loss: 0.5788 - val_accuracy: 0.7420
Epoch 9/15
313/313 [==============================] - 176s 562ms/step - loss: 0.4679 -
accuracy: 0.7779 - val_loss: 0.5971 - val_accuracy: 0.7330
Epoch 10/15
313/313 [==============================] - 176s 564ms/step - loss: 0.4640 -
accuracy: 0.7781 - val_loss: 0.6606 - val_accuracy: 0.6060
Epoch 11/15
313/313 [==============================] - 179s 571ms/step - loss: 0.5244 -
accuracy: 0.7294 - val_loss: 0.9755 - val_accuracy: 0.5750
Epoch 12/15
313/313 [==============================] - 180s 576ms/step - loss: 0.5606 -
accuracy: 0.7060 - val_loss: 0.6358 - val_accuracy: 0.6560
Epoch 13/15
313/313 [==============================] - 179s 571ms/step - loss: 0.4908 -
accuracy: 0.7535 - val_loss: 0.6270 - val_accuracy: 0.6804
Epoch 14/15
313/313 [==============================] - 170s 544ms/step - loss: 0.4166 -
accuracy: 0.8142 - val_loss: 0.6113 - val_accuracy: 0.7260
Epoch 15/15
313/313 [==============================] - 176s 564ms/step - loss: 0.3945 -
accuracy: 0.8316 - val_loss: 0.6510 - val_accuracy: 0.7116
```

[8]:
```
model.evaluate(X_test,y_test)
```

```
782/782 [==============================] - 34s 44ms/step - loss: 0.6362 -
accuracy: 0.7154
```

[8]:
```
[0.6362121105194092, 0.7153599858283997]
```

6 8. Analyse and visualize the performance change while using LSTM and GRU instead of simple RNN

[1]:
```
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing import sequence
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, LSTM, GRU, Dense

# Load the IMDB dataset
```

```python
max_features = 10000  # Number of words to consider as features
maxlen = 500  # Cut off reviews after this number of words
batch_size = 32

print('Loading data...')
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words=max_features)
print(len(train_data), 'train sequences')
print(len(test_data), 'test sequences')

# Pad sequences to a fixed length
print('Pad sequences (samples x time)')
train_data = sequence.pad_sequences(train_data, maxlen=maxlen)
test_data = sequence.pad_sequences(test_data, maxlen=maxlen)
print('Train data shape:', train_data.shape)
print('Test data shape:', test_data.shape)

# Define a function to create and train a model
def create_and_train_model(model_type):
    model = Sequential()

    # Add an Embedding layer
    model.add(Embedding(max_features, 32))

    # Choose the RNN layer based on the provided model type
    if model_type == 'SimpleRNN':
        model.add(SimpleRNN(32))
    elif model_type == 'LSTM':
        model.add(LSTM(32))
    elif model_type == 'GRU':
        model.add(GRU(32))
    else:
        raise ValueError("Invalid model type. Use 'SimpleRNN', 'LSTM', or 'GRU'.")

    # Add a Dense layer
    model.add(Dense(1, activation='sigmoid'))

    # Compile the model
    model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy'])

    # Train the model
    history = model.fit(train_data, train_labels, epochs=5, batch_size=batch_size, validation_split=0.2, verbose=0)

    return model, history
```

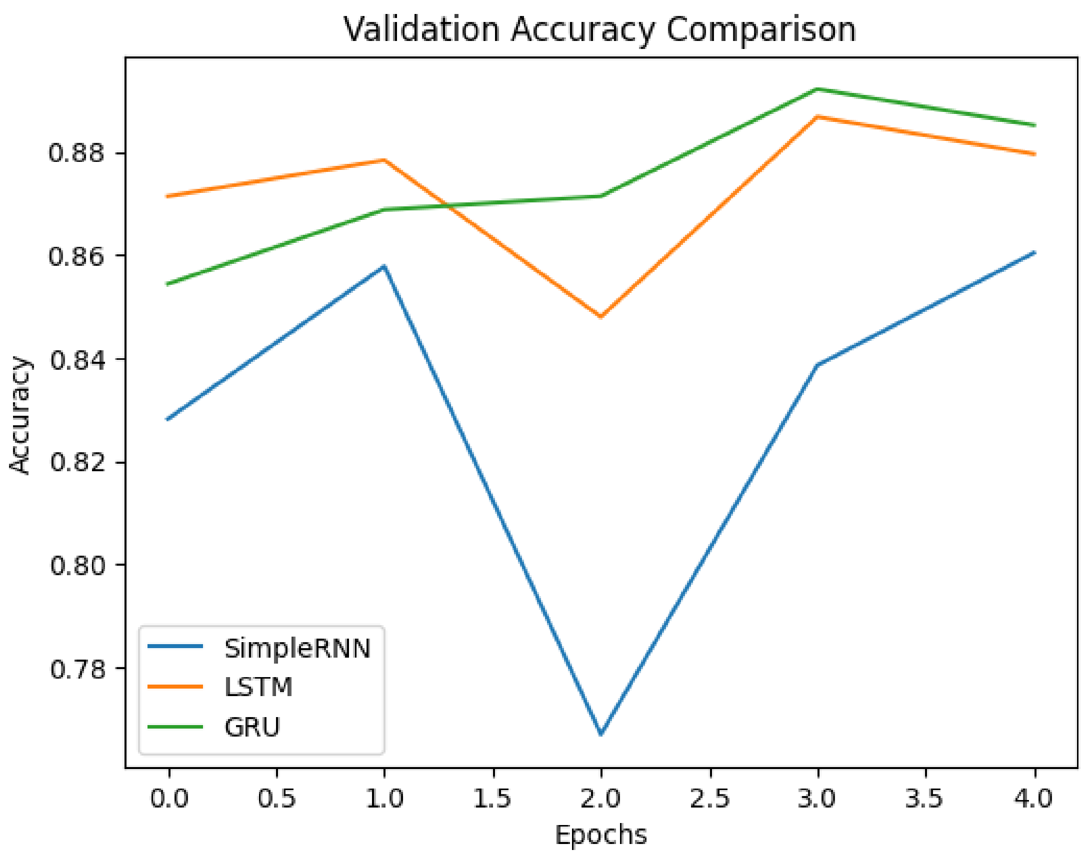
```
# Create and train models for SimpleRNN, LSTM, and GRU
model_rnn, history_rnn = create_and_train_model('SimpleRNN')
model_lstm, history_lstm = create_and_train_model('LSTM')
model_gru, history_gru = create_and_train_model('GRU')

# Evaluate models on the test set
results_rnn = model_rnn.evaluate(test_data, test_labels, verbose=0)
results_lstm = model_lstm.evaluate(test_data, test_labels, verbose=0)
results_gru = model_gru.evaluate(test_data, test_labels, verbose=0)

# Print test accuracy
print(f'Test accuracy (SimpleRNN): {results_rnn[1]}')
print(f'Test accuracy (LSTM): {results_lstm[1]}')
print(f'Test accuracy (GRU): {results_gru[1]}')
```

```
Loading data…
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/imdb.npz
17464789/17464789 [==============================] - 0s 0us/step
25000 train sequences
25000 test sequences
Pad sequences (samples x time)
Train data shape: (25000, 500)
Test data shape: (25000, 500)
Test accuracy (SimpleRNN): 0.8595200181007385
Test accuracy (LSTM): 0.8720399737358093
Test accuracy (GRU): 0.8795599937438965
```

[2]:
```
# Plot validation accuracy
plt.plot(history_rnn.history['val_accuracy'], label='SimpleRNN')
plt.plot(history_lstm.history['val_accuracy'], label='LSTM')
plt.plot(history_gru.history['val_accuracy'], label='GRU')
plt.title('Validation Accuracy Comparison')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



7 9. Implement time series forecasting prediction for NIFTY-50 dataset.

[4]:
```
import pandas as pd
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, SimpleRNN
from keras import layers
from sklearn.preprocessing import MinMaxScaler
from keras.preprocessing.sequence import TimeseriesGenerator
import matplotlib.pyplot as plt

# Load the dataset
data = pd.read_csv('/content/drive/MyDrive/DL LAB S7/NIFTY.csv',␣
 ↪index_col='Date', parse_dates=True)
data.head()
```

```
<ipython-input-4-bbf3a80c323a>:11: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
```

```
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  data = pd.read_csv('/content/drive/MyDrive/DL LAB S7/NIFTY.csv',
index_col='Date', parse_dates=True)
```

[4]:

```
             Open   High    Low  Turnover
Date
2009-02-03  43.19  43.38  41.44     43.17
2009-03-03  43.17  43.90  41.20     43.89
2009-04-03  43.89  43.89  42.16     42.52
2009-05-03  42.52  42.71  40.41     41.49
2009-06-03  41.49  41.49  37.57     38.16
```

[5]:

```
# Normalize the data
scaler = MinMaxScaler(feature_range=(0, 1))
data_scaled = scaler.fit_transform(data)
data_scaled
```

[5]:

```
array([[0.44754647, 0.42956052, 0.4862573 , 0.4472731 ],
       [0.4472731 , 0.43641819, 0.4826868 , 0.45711454],
       [0.45711454, 0.43628631, 0.4969688 , 0.43838846],
       …,
       [0.45325314, 0.43285747, 0.46777253, 0.42099508],
       [0.42099508, 0.40760278, 0.45973891, 0.4029866 ],
       [0.4029866 , 0.38406251, 0.41347119, 0.38217605]])
```

[8]:

```
# Split the data into training and testing sets
n = int(len(data_scaled) * 0.8)
train_data = data_scaled[:n]
test_data = data_scaled[n:]

# Define the parameters
n_input = 5
n_features = 4
```

[9]:

```
# Create time series generators
generator_train = TimeseriesGenerator(train_data, train_data, length=n_input)
generator_test = TimeseriesGenerator(test_data, test_data, length=n_input)
```

[12]:

```
# Build the RNN model
model = Sequential()
model.add(layers.LSTM(50, activation='relu', input_shape=(n_input, n_features)))
model.add(Dense(4))
model.compile(optimizer='adam',
              loss='mean_squared_error',
              metrics=['accuracy'])

# Display the model summary
```

```python
print(model.summary())
```

```
Model: "sequential_3"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_1 (LSTM)               (None, 50)                11000

 dense_3 (Dense)             (None, 4)                 204

=================================================================
Total params: 11204 (43.77 KB)
Trainable params: 11204 (43.77 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

[13]:

```python
# Train the model
model.fit(generator_train, epochs=50)
```

```
Epoch 1/50
18/18 [==============================] - 2s 11ms/step - loss: 0.0185 - accuracy:
0.1629
Epoch 2/50
18/18 [==============================] - 0s 11ms/step - loss: 0.0092 - accuracy:
0.7353
Epoch 3/50
18/18 [==============================] - 0s 12ms/step - loss: 0.0041 - accuracy:
0.8208
Epoch 4/50
18/18 [==============================] - 0s 12ms/step - loss: 0.0025 - accuracy:
0.8208
Epoch 5/50
18/18 [==============================] - 0s 11ms/step - loss: 0.0016 - accuracy:
0.8208
Epoch 6/50
18/18 [==============================] - 0s 11ms/step - loss: 8.6639e-04 -
accuracy: 0.7308
Epoch 7/50
18/18 [==============================] - 0s 11ms/step - loss: 6.5189e-04 -
accuracy: 0.7729
Epoch 8/50
18/18 [==============================] - 0s 11ms/step - loss: 5.7697e-04 -
accuracy: 0.8167
Epoch 9/50
18/18 [==============================] - 0s 12ms/step - loss: 5.5013e-04 -
accuracy: 0.8213
Epoch 10/50
```

```
18/18 [==============================] - 0s 12ms/step - loss: 5.3958e-04 -
accuracy: 0.8208
Epoch 11/50
18/18 [==============================] - 0s 13ms/step - loss: 5.2835e-04 -
accuracy: 0.8213
Epoch 12/50
18/18 [==============================] - 0s 15ms/step - loss: 5.2390e-04 -
accuracy: 0.8204
Epoch 13/50
18/18 [==============================] - 0s 15ms/step - loss: 5.2165e-04 -
accuracy: 0.8208
Epoch 14/50
18/18 [==============================] - 0s 13ms/step - loss: 5.1137e-04 -
accuracy: 0.8213
Epoch 15/50
18/18 [==============================] - 0s 11ms/step - loss: 5.0635e-04 -
accuracy: 0.8213
Epoch 16/50
18/18 [==============================] - 0s 7ms/step - loss: 4.9948e-04 -
accuracy: 0.8213
Epoch 17/50
18/18 [==============================] - 0s 8ms/step - loss: 5.0324e-04 -
accuracy: 0.8213
Epoch 18/50
18/18 [==============================] - 0s 7ms/step - loss: 4.9239e-04 -
accuracy: 0.8208
Epoch 19/50
18/18 [==============================] - 0s 7ms/step - loss: 4.8335e-04 -
accuracy: 0.8213
Epoch 20/50
18/18 [==============================] - 0s 8ms/step - loss: 4.7227e-04 -
accuracy: 0.8213
Epoch 21/50
18/18 [==============================] - 0s 8ms/step - loss: 4.6259e-04 -
accuracy: 0.8213
Epoch 22/50
18/18 [==============================] - 0s 8ms/step - loss: 4.5924e-04 -
accuracy: 0.8213
Epoch 23/50
18/18 [==============================] - 0s 7ms/step - loss: 4.4761e-04 -
accuracy: 0.8208
Epoch 24/50
18/18 [==============================] - 0s 8ms/step - loss: 4.3659e-04 -
accuracy: 0.8213
Epoch 25/50
18/18 [==============================] - 0s 7ms/step - loss: 4.2622e-04 -
accuracy: 0.8208
Epoch 26/50
```

```
18/18 [==============================] - 0s 7ms/step - loss: 4.2085e-04 -
accuracy: 0.8208
Epoch 27/50
18/18 [==============================] - 0s 7ms/step - loss: 4.0638e-04 -
accuracy: 0.8213
Epoch 28/50
18/18 [==============================] - 0s 7ms/step - loss: 3.9785e-04 -
accuracy: 0.8213
Epoch 29/50
18/18 [==============================] - 0s 7ms/step - loss: 3.9060e-04 -
accuracy: 0.8208
Epoch 30/50
18/18 [==============================] - 0s 7ms/step - loss: 3.9169e-04 -
accuracy: 0.8213
Epoch 31/50
18/18 [==============================] - 0s 7ms/step - loss: 3.6812e-04 -
accuracy: 0.8208
Epoch 32/50
18/18 [==============================] - 0s 8ms/step - loss: 3.6693e-04 -
accuracy: 0.8213
Epoch 33/50
18/18 [==============================] - 0s 7ms/step - loss: 3.5517e-04 -
accuracy: 0.8208
Epoch 34/50
18/18 [==============================] - 0s 7ms/step - loss: 3.7218e-04 -
accuracy: 0.8208
Epoch 35/50
18/18 [==============================] - 0s 8ms/step - loss: 3.5928e-04 -
accuracy: 0.8208
Epoch 36/50
18/18 [==============================] - 0s 8ms/step - loss: 3.3628e-04 -
accuracy: 0.8208
Epoch 37/50
18/18 [==============================] - 0s 8ms/step - loss: 3.3466e-04 -
accuracy: 0.8217
Epoch 38/50
18/18 [==============================] - 0s 8ms/step - loss: 3.5476e-04 -
accuracy: 0.8213
Epoch 39/50
18/18 [==============================] - 0s 8ms/step - loss: 3.4707e-04 -
accuracy: 0.8208
Epoch 40/50
18/18 [==============================] - 0s 8ms/step - loss: 3.9313e-04 -
accuracy: 0.8181
Epoch 41/50
18/18 [==============================] - 0s 8ms/step - loss: 3.5195e-04 -
accuracy: 0.8176
Epoch 42/50
```

```
18/18 [==============================] - 0s 8ms/step - loss: 3.2687e-04 -
accuracy: 0.8217
Epoch 43/50
18/18 [==============================] - 0s 8ms/step - loss: 3.1042e-04 -
accuracy: 0.8213
Epoch 44/50
18/18 [==============================] - 0s 7ms/step - loss: 3.1350e-04 -
accuracy: 0.8163
Epoch 45/50
18/18 [==============================] - 0s 7ms/step - loss: 3.0630e-04 -
accuracy: 0.8217
Epoch 46/50
18/18 [==============================] - 0s 8ms/step - loss: 2.9716e-04 -
accuracy: 0.8208
Epoch 47/50
18/18 [==============================] - 0s 7ms/step - loss: 2.9350e-04 -
accuracy: 0.8208
Epoch 48/50
18/18 [==============================] - 0s 8ms/step - loss: 2.9266e-04 -
accuracy: 0.8208
Epoch 49/50
18/18 [==============================] - 0s 8ms/step - loss: 2.9128e-04 -
accuracy: 0.8217
Epoch 50/50
18/18 [==============================] - 0s 8ms/step - loss: 2.8891e-04 -
accuracy: 0.8217
```

[13]:
```
<keras.src.callbacks.History at 0x795af86391b0>
```

[14]:
```
# Evaluate the model on the test set
test_loss = model.evaluate(generator_test)
print(f'Test Loss: {test_loss}')
```

```
5/5 [==============================] - 0s 7ms/step - loss: 8.0705e-04 -
accuracy: 0.5683
Test Loss: [0.0008070533513091505, 0.568306028842926]
```

[15]:
```
# Make predictions on the test set
predictions = model.predict(generator_test)
predictions
```

```
5/5 [==============================] - 1s 8ms/step
```

[15]:
```
array([[0.11884815, 0.12556748, 0.13174699, 0.118283  ],
       [0.10907926, 0.11520067, 0.12524894, 0.10981216],
       [0.09840428, 0.1026832 , 0.11601608, 0.09911459],
       …,
       [0.4169026 , 0.42017323, 0.4349164 , 0.40776107],
```

```
        [0.41951257, 0.41825992, 0.44075045, 0.4099289 ],
        [0.4132465 , 0.4095242 , 0.43619075, 0.4024378 ]], dtype=float32)
```

[16]:
```
# Inverse transform the predictions and actual values to the original scale
predictions_original = scaler.inverse_transform(predictions)
test_data_original = scaler.inverse_transform(test_data[n_input:])
test_data_original
```

[16]:
```
array([[19.23   , 19.23   , 16.5675, 17.8825],
       [17.8825, 17.8825, 15.52   , 17.185 ],
       [17.185 , 17.185 , 14.085 , 16.315 ],
       …,
       [43.6075, 43.63   , 40.1975, 41.2475],
       [41.2475, 41.715 , 39.6575, 39.93  ],
       [39.93   , 39.93   , 36.5475, 38.4075]])
```

[17]:
```
# Plot the results
plt.figure(figsize=(15, 6))
plt.plot(data.index[n + n_input:], test_data_original, label='Actual Prices',␣
 ↪color='blue')
plt.plot(data.index[n + n_input:], predictions_original, label='Predicted␣
 ↪Prices', color='orange')
plt.title('NIFTY-50 Stock Price Prediction using RNN')
plt.xlabel('Date')
plt.ylabel('Stock Price (Close)')
plt.legend()
plt.show()
```