dl-lab-experiments-2

December 14, 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
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
```

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

[ ]:
```
print(f'X_train shape: {X_train.shape}\ny_train shape: {y_train.shape}')
print(f'X_test shape: {X_test.shape}\ny_test shape: {y_test.shape}')
```

```
X_train shape: (50000, 32, 32, 3)
y_train shape: (50000, 1)
X_test shape: (10000, 32, 32, 3)
y_test shape: (10000, 1)
```

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

[ ]:
```
print(f'X_train shape: {X_train.shape}\ny_train shape: {y_train.shape}')
print(f'X_test shape: {X_test.shape}\ny_test shape: {y_test.shape}')
```

```
X_train shape: (50000, 32, 32, 3)
y_train shape: (50000, 10)
X_test shape: (10000, 32, 32, 3)
y_test shape: (10000, 10)
```

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

```python
# 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'))
```

[ ]:
```python
# 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(X_train, y_train, epochs=15, validation_data=(X_test,␣
 ↪y_test))
```

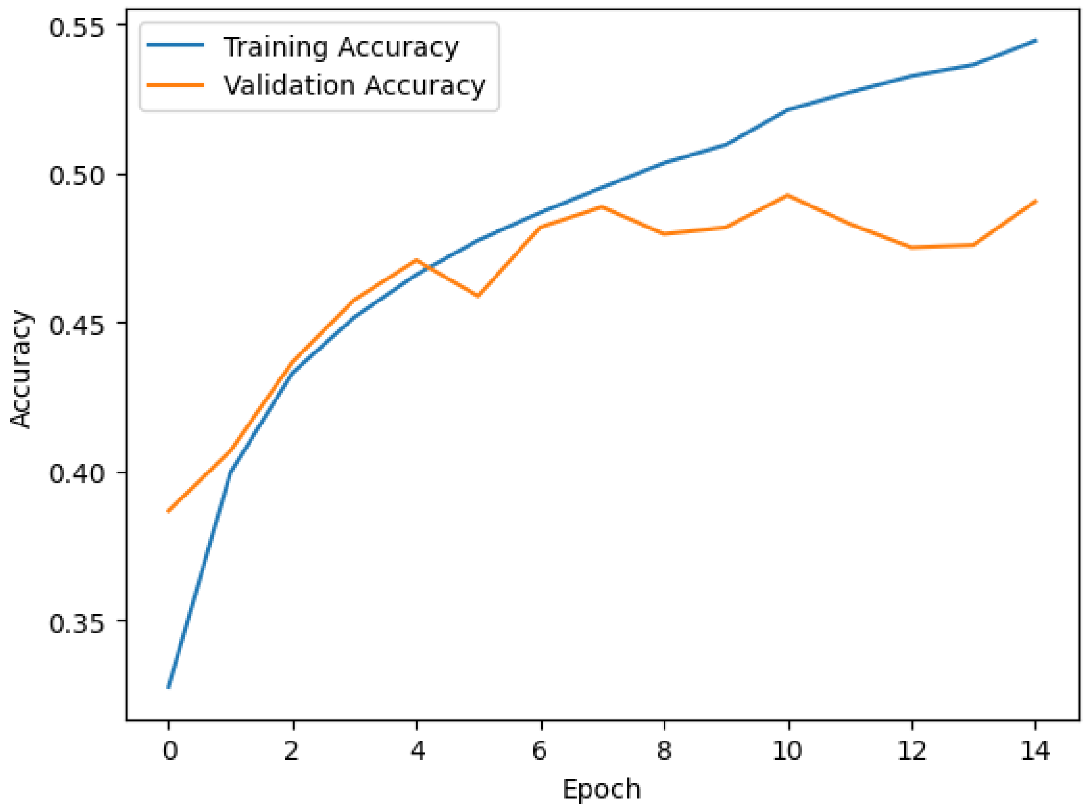
```
Epoch 1/15
1563/1563 [==============================] - 13s 5ms/step - loss: 1.8592 -
accuracy: 0.3278 - val_loss: 1.6996 - val_accuracy: 0.3868
Epoch 2/15
```

```
1563/1563 [==============================] - 7s 4ms/step - loss: 1.6749 -
accuracy: 0.3996 - val_loss: 1.6449 - val_accuracy: 0.4069
Epoch 3/15
1563/1563 [==============================] - 7s 5ms/step - loss: 1.5880 -
accuracy: 0.4330 - val_loss: 1.5731 - val_accuracy: 0.4367
Epoch 4/15
1563/1563 [==============================] - 7s 4ms/step - loss: 1.5302 -
accuracy: 0.4517 - val_loss: 1.5233 - val_accuracy: 0.4575
Epoch 5/15
1563/1563 [==============================] - 6s 4ms/step - loss: 1.4919 -
accuracy: 0.4660 - val_loss: 1.5045 - val_accuracy: 0.4708
Epoch 6/15
1563/1563 [==============================] - 7s 5ms/step - loss: 1.4612 -
accuracy: 0.4775 - val_loss: 1.5188 - val_accuracy: 0.4588
Epoch 7/15
1563/1563 [==============================] - 7s 4ms/step - loss: 1.4365 -
accuracy: 0.4868 - val_loss: 1.4675 - val_accuracy: 0.4818
Epoch 8/15
1563/1563 [==============================] - 7s 4ms/step - loss: 1.4100 -
accuracy: 0.4952 - val_loss: 1.4518 - val_accuracy: 0.4887
Epoch 9/15
1563/1563 [==============================] - 7s 4ms/step - loss: 1.3890 -
accuracy: 0.5034 - val_loss: 1.4769 - val_accuracy: 0.4796
Epoch 10/15
1563/1563 [==============================] - 7s 5ms/step - loss: 1.3650 -
accuracy: 0.5095 - val_loss: 1.4687 - val_accuracy: 0.4818
Epoch 11/15
1563/1563 [==============================] - 11s 7ms/step - loss: 1.3455 -
accuracy: 0.5212 - val_loss: 1.4320 - val_accuracy: 0.4926
Epoch 12/15
1563/1563 [==============================] - 9s 6ms/step - loss: 1.3239 -
accuracy: 0.5271 - val_loss: 1.4626 - val_accuracy: 0.4830
Epoch 13/15
1563/1563 [==============================] - 8s 5ms/step - loss: 1.3052 -
accuracy: 0.5326 - val_loss: 1.5024 - val_accuracy: 0.4751
Epoch 14/15
1563/1563 [==============================] - 7s 4ms/step - loss: 1.2937 -
accuracy: 0.5364 - val_loss: 1.4824 - val_accuracy: 0.4759
Epoch 15/15
1563/1563 [==============================] - 6s 4ms/step - loss: 1.2708 -
accuracy: 0.5444 - val_loss: 1.4649 - val_accuracy: 0.4905
```

[ ]:
```
# Evaluate the model
score = model.evaluate(X_test, y_test)
```

```
313/313 [==============================] - 1s 3ms/step - loss: 1.4649 -
accuracy: 0.4905
```

[ ]:
```
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()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
```

```
170498071/170498071 [==============================] - 6s 0us/step
```

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

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

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

[ ]:
```python
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'))
```

```
[ ]: # 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'))
```

```
[ ]: sgd_optimizer = optimizers.SGD(learning_rate=0.01, momentum=0.9)
model1.compile(optimizer=sgd_optimizer,
              loss='categorical_crossentropy',
              metrics=['accuracy'])
print(model1.summary())
Xavier_history  = model1.
 ↪fit(X_train,y_train,epochs=15,batch_size=32,validation_split=0.2)
Xavier_score = model1.evaluate(X_test,y_test,batch_size=32)
print(Xavier_score)
```

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

 dense (Dense)               (None, 256)               786688

 dense_1 (Dense)             (None, 256)               65792

 dense_2 (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 [==============================] - 11s 5ms/step - loss: 1.8755 -
accuracy: 0.3203 - val_loss: 1.8879 - val_accuracy: 0.3179
Epoch 2/15
1250/1250 [==============================] - 7s 6ms/step - loss: 1.7116 -
accuracy: 0.3821 - val_loss: 1.7003 - val_accuracy: 0.3947
Epoch 3/15
1250/1250 [==============================] - 5s 4ms/step - loss: 1.6399 -
accuracy: 0.4110 - val_loss: 1.6479 - val_accuracy: 0.4036
Epoch 4/15
1250/1250 [==============================] - 7s 6ms/step - loss: 1.5987 -
```

```
accuracy: 0.4257 - val_loss: 1.6595 - val_accuracy: 0.4075
Epoch 5/15
1250/1250 [==============================] - 6s 5ms/step - loss: 1.5721 -
accuracy: 0.4345 - val_loss: 1.6283 - val_accuracy: 0.4223
Epoch 6/15
1250/1250 [==============================] - 6s 5ms/step - loss: 1.5427 -
accuracy: 0.4479 - val_loss: 1.6270 - val_accuracy: 0.4249
Epoch 7/15
1250/1250 [==============================] - 8s 6ms/step - loss: 1.5200 -
accuracy: 0.4543 - val_loss: 1.6115 - val_accuracy: 0.4251
Epoch 8/15
1250/1250 [==============================] - 8s 6ms/step - loss: 1.5047 -
accuracy: 0.4614 - val_loss: 1.5817 - val_accuracy: 0.4326
Epoch 9/15
1250/1250 [==============================] - 5s 4ms/step - loss: 1.4787 -
accuracy: 0.4676 - val_loss: 1.6318 - val_accuracy: 0.4289
Epoch 10/15
1250/1250 [==============================] - 6s 5ms/step - loss: 1.4709 -
accuracy: 0.4707 - val_loss: 1.5859 - val_accuracy: 0.4364
Epoch 11/15
1250/1250 [==============================] - 7s 6ms/step - loss: 1.4467 -
accuracy: 0.4812 - val_loss: 1.6028 - val_accuracy: 0.4405
Epoch 12/15
1250/1250 [==============================] - 6s 5ms/step - loss: 1.4289 -
accuracy: 0.4861 - val_loss: 1.5789 - val_accuracy: 0.4499
Epoch 13/15
1250/1250 [==============================] - 6s 5ms/step - loss: 1.4155 -
accuracy: 0.4927 - val_loss: 1.5831 - val_accuracy: 0.4351
Epoch 14/15
1250/1250 [==============================] - 4s 4ms/step - loss: 1.4029 -
accuracy: 0.4967 - val_loss: 1.5780 - val_accuracy: 0.4483
Epoch 15/15
1250/1250 [==============================] - 6s 5ms/step - loss: 1.3904 -
accuracy: 0.5012 - val_loss: 1.5149 - val_accuracy: 0.4675
313/313 [==============================] - 1s 3ms/step - loss: 1.5040 -
accuracy: 0.4652
[1.5040026903152466, 0.4652000069618225]
```

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

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_1 (Flatten)         (None, 3072)              0

 dense_3 (Dense)             (None, 256)               786688

 dense_4 (Dense)             (None, 128)               32896

 dense_5 (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 5ms/step - loss: 1.8932 -
accuracy: 0.3129 - val_loss: 1.8377 - val_accuracy: 0.3323
Epoch 2/15
1250/1250 [==============================] - 5s 4ms/step - loss: 1.7308 -
accuracy: 0.3795 - val_loss: 1.7861 - val_accuracy: 0.3530
Epoch 3/15
1250/1250 [==============================] - 5s 4ms/step - loss: 1.6724 -
accuracy: 0.4001 - val_loss: 1.6722 - val_accuracy: 0.3971
Epoch 4/15
1250/1250 [==============================] - 7s 5ms/step - loss: 1.6274 -
accuracy: 0.4169 - val_loss: 1.6683 - val_accuracy: 0.4111
Epoch 5/15
1250/1250 [==============================] - 6s 4ms/step - loss: 1.5974 -
accuracy: 0.4286 - val_loss: 1.6234 - val_accuracy: 0.4245
Epoch 6/15
1250/1250 [==============================] - 6s 5ms/step - loss: 1.5605 -
accuracy: 0.4411 - val_loss: 1.6466 - val_accuracy: 0.4192
Epoch 7/15
1250/1250 [==============================] - 5s 4ms/step - loss: 1.5411 -
accuracy: 0.4503 - val_loss: 1.5860 - val_accuracy: 0.4396
Epoch 8/15
1250/1250 [==============================] - 5s 4ms/step - loss: 1.5239 -
accuracy: 0.4536 - val_loss: 1.7091 - val_accuracy: 0.4008
Epoch 9/15
1250/1250 [==============================] - 7s 5ms/step - loss: 1.5152 -
accuracy: 0.4567 - val_loss: 1.5768 - val_accuracy: 0.4461
Epoch 10/15
1250/1250 [==============================] - 10s 8ms/step - loss: 1.4989 -
accuracy: 0.4622 - val_loss: 1.5809 - val_accuracy: 0.4467
```

```
Epoch 11/15
1250/1250 [==============================] - 7s 6ms/step - loss: 1.4892 -
accuracy: 0.4690 - val_loss: 1.5822 - val_accuracy: 0.4462
Epoch 12/15
1250/1250 [==============================] - 7s 6ms/step - loss: 1.4619 -
accuracy: 0.4761 - val_loss: 1.5610 - val_accuracy: 0.4498
Epoch 13/15
1250/1250 [==============================] - 5s 4ms/step - loss: 1.4540 -
accuracy: 0.4814 - val_loss: 1.5693 - val_accuracy: 0.4469
Epoch 14/15
1250/1250 [==============================] - 6s 5ms/step - loss: 1.4407 -
accuracy: 0.4827 - val_loss: 1.5491 - val_accuracy: 0.4594
Epoch 15/15
1250/1250 [==============================] - 5s 4ms/step - loss: 1.4353 -
accuracy: 0.4843 - val_loss: 1.5748 - val_accuracy: 0.4488
79/79 [==============================] - 0s 3ms/step - loss: 1.5467 - accuracy:
0.4579
[1.54667329788208, 0.4578999876976013]
```

[ ]:
```python
sgd_optimizer = optimizers.SGD(learning_rate=0.01, momentum=0.9)
model3.compile(optimizer=sgd_optimizer,
              loss='categorical_crossentropy',
              metrics=['accuracy'])
print(model3.summary())

dropout_history  = model3.
 ↪fit(X_train,y_train,epochs=15,batch_size=32,validation_split=0.2)
dropout_score = model3.evaluate(X_test,y_test,batch_size=128)
print(dropout_score)
```

```
Model: "sequential_2"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_2 (Flatten)         (None, 3072)              0

 dense_6 (Dense)             (None, 256)               786688

 dropout (Dropout)           (None, 256)               0

 dense_7 (Dense)             (None, 128)               32896

 dense_8 (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 [==============================] - 9s 5ms/step - loss: 2.0002 -
accuracy: 0.2613 - val_loss: 1.8789 - val_accuracy: 0.3180
Epoch 2/15
1250/1250 [==============================] - 5s 4ms/step - loss: 1.8879 -
accuracy: 0.3088 - val_loss: 1.8183 - val_accuracy: 0.3398
Epoch 3/15
1250/1250 [==============================] - 7s 6ms/step - loss: 1.8400 -
accuracy: 0.3270 - val_loss: 1.7877 - val_accuracy: 0.3499
Epoch 4/15
1250/1250 [==============================] - 5s 4ms/step - loss: 1.7914 -
accuracy: 0.3453 - val_loss: 1.7410 - val_accuracy: 0.3655
Epoch 5/15
1250/1250 [==============================] - 5s 4ms/step - loss: 1.7679 -
accuracy: 0.3561 - val_loss: 1.7466 - val_accuracy: 0.3774
Epoch 6/15
1250/1250 [==============================] - 5s 4ms/step - loss: 1.7416 -
accuracy: 0.3689 - val_loss: 1.6831 - val_accuracy: 0.3946
Epoch 7/15
1250/1250 [==============================] - 5s 4ms/step - loss: 1.7284 -
accuracy: 0.3720 - val_loss: 1.6930 - val_accuracy: 0.3958
Epoch 8/15
1250/1250 [==============================] - 6s 5ms/step - loss: 1.7120 -
accuracy: 0.3805 - val_loss: 1.6957 - val_accuracy: 0.3885
Epoch 9/15
1250/1250 [==============================] - 5s 4ms/step - loss: 1.7019 -
accuracy: 0.3849 - val_loss: 1.6806 - val_accuracy: 0.4138
Epoch 10/15
1250/1250 [==============================] - 5s 4ms/step - loss: 1.6828 -
accuracy: 0.3924 - val_loss: 1.6229 - val_accuracy: 0.4289
Epoch 11/15
1250/1250 [==============================] - 5s 4ms/step - loss: 1.6775 -
accuracy: 0.3933 - val_loss: 1.6534 - val_accuracy: 0.4150
Epoch 12/15
1250/1250 [==============================] - 6s 4ms/step - loss: 1.6607 -
accuracy: 0.3996 - val_loss: 1.6522 - val_accuracy: 0.4134
Epoch 13/15
1250/1250 [==============================] - 5s 4ms/step - loss: 1.6433 -
accuracy: 0.4073 - val_loss: 1.6365 - val_accuracy: 0.4357
Epoch 14/15
1250/1250 [==============================] - 5s 4ms/step - loss: 1.6384 -
accuracy: 0.4091 - val_loss: 1.6059 - val_accuracy: 0.4261
Epoch 15/15
1250/1250 [==============================] - 5s 4ms/step - loss: 1.6356 -
accuracy: 0.4095 - val_loss: 1.6004 - val_accuracy: 0.4362
79/79 [==============================] - 0s 5ms/step - loss: 1.5753 - accuracy:
```

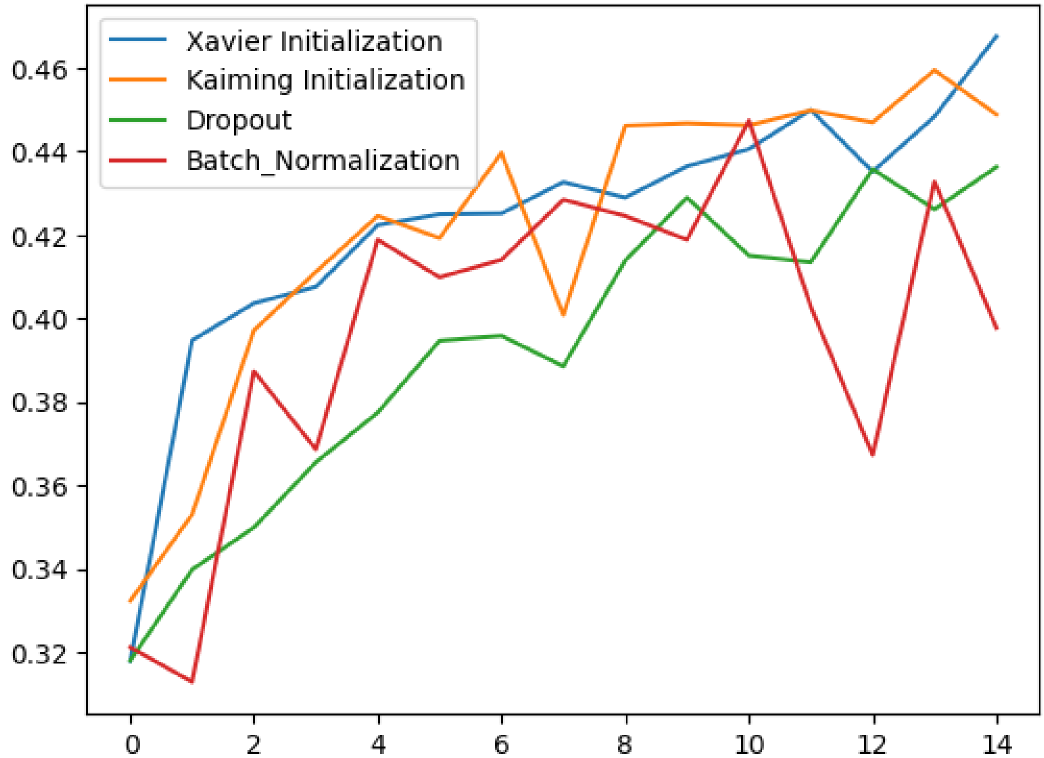
```
0.4425
[1.575345754623413, 0.4424999952316284]
```

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

BN_history = model4.
 ↪fit(X_train,y_train,epochs=15,batch_size=128,validation_split=0.2)
BN_score = model4.evaluate(X_test,y_test,batch_size=128)
print(BN_score)
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_3 (Flatten)         (None, 3072)              0

 dense_9 (Dense)             (None, 256)               786688

 batch_normalization (Batch  (None, 256)               1024
 Normalization)

 activation (Activation)     (None, 256)               0

 dense_10 (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: 1.7435 -
accuracy: 0.3929 - val_loss: 1.9077 - val_accuracy: 0.3212
Epoch 2/15
313/313 [==============================] - 3s 9ms/step - loss: 1.5787 -
accuracy: 0.4527 - val_loss: 2.0264 - val_accuracy: 0.3129
Epoch 3/15
313/313 [==============================] - 2s 8ms/step - loss: 1.4976 -
accuracy: 0.4791 - val_loss: 1.7964 - val_accuracy: 0.3873
Epoch 4/15
313/313 [==============================] - 2s 5ms/step - loss: 1.4604 -
accuracy: 0.4917 - val_loss: 1.7981 - val_accuracy: 0.3686
Epoch 5/15
```

```
313/313 [==============================] - 2s 5ms/step - loss: 1.4433 -
accuracy: 0.4959 - val_loss: 1.6776 - val_accuracy: 0.4188
Epoch 6/15
313/313 [==============================] - 2s 5ms/step - loss: 1.4147 -
accuracy: 0.5046 - val_loss: 1.6728 - val_accuracy: 0.4098
Epoch 7/15
313/313 [==============================] - 2s 8ms/step - loss: 1.3822 -
accuracy: 0.5164 - val_loss: 1.6894 - val_accuracy: 0.4140
Epoch 8/15
313/313 [==============================] - 2s 8ms/step - loss: 1.3502 -
accuracy: 0.5282 - val_loss: 1.6909 - val_accuracy: 0.4284
Epoch 9/15
313/313 [==============================] - 4s 11ms/step - loss: 1.3272 -
accuracy: 0.5384 - val_loss: 1.6696 - val_accuracy: 0.4245
Epoch 10/15
313/313 [==============================] - 2s 6ms/step - loss: 1.3183 -
accuracy: 0.5387 - val_loss: 1.7419 - val_accuracy: 0.4188
Epoch 11/15
313/313 [==============================] - 2s 5ms/step - loss: 1.2873 -
accuracy: 0.5481 - val_loss: 1.6096 - val_accuracy: 0.4474
Epoch 12/15
313/313 [==============================] - 2s 5ms/step - loss: 1.2667 -
accuracy: 0.5526 - val_loss: 1.8200 - val_accuracy: 0.4028
Epoch 13/15
313/313 [==============================] - 2s 5ms/step - loss: 1.2532 -
accuracy: 0.5611 - val_loss: 2.0636 - val_accuracy: 0.3672
Epoch 14/15
313/313 [==============================] - 2s 5ms/step - loss: 1.2398 -
accuracy: 0.5660 - val_loss: 1.7374 - val_accuracy: 0.4327
Epoch 15/15
313/313 [==============================] - 2s 6ms/step - loss: 1.2253 -
accuracy: 0.5719 - val_loss: 1.8646 - val_accuracy: 0.3977
79/79 [==============================] - 0s 4ms/step - loss: 1.8478 - accuracy:
0.4013
[1.8477566242218018, 0.40130001306533813]
```

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

plt.plot(Xavier_history.history['val_accuracy'],label='Xavier Initialization')
plt.plot(Kaiming_history.history['val_accuracy'],label='Kaiming Initialization')
plt.plot(dropout_history.history['val_accuracy'],label='Dropout')
plt.plot(BN_history.history['val_accuracy'],label='Batch_Normalization')
plt.legend()
plt.show()
```



3 5. Digit Classification using CNN Architecture for MNIST Dataset

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

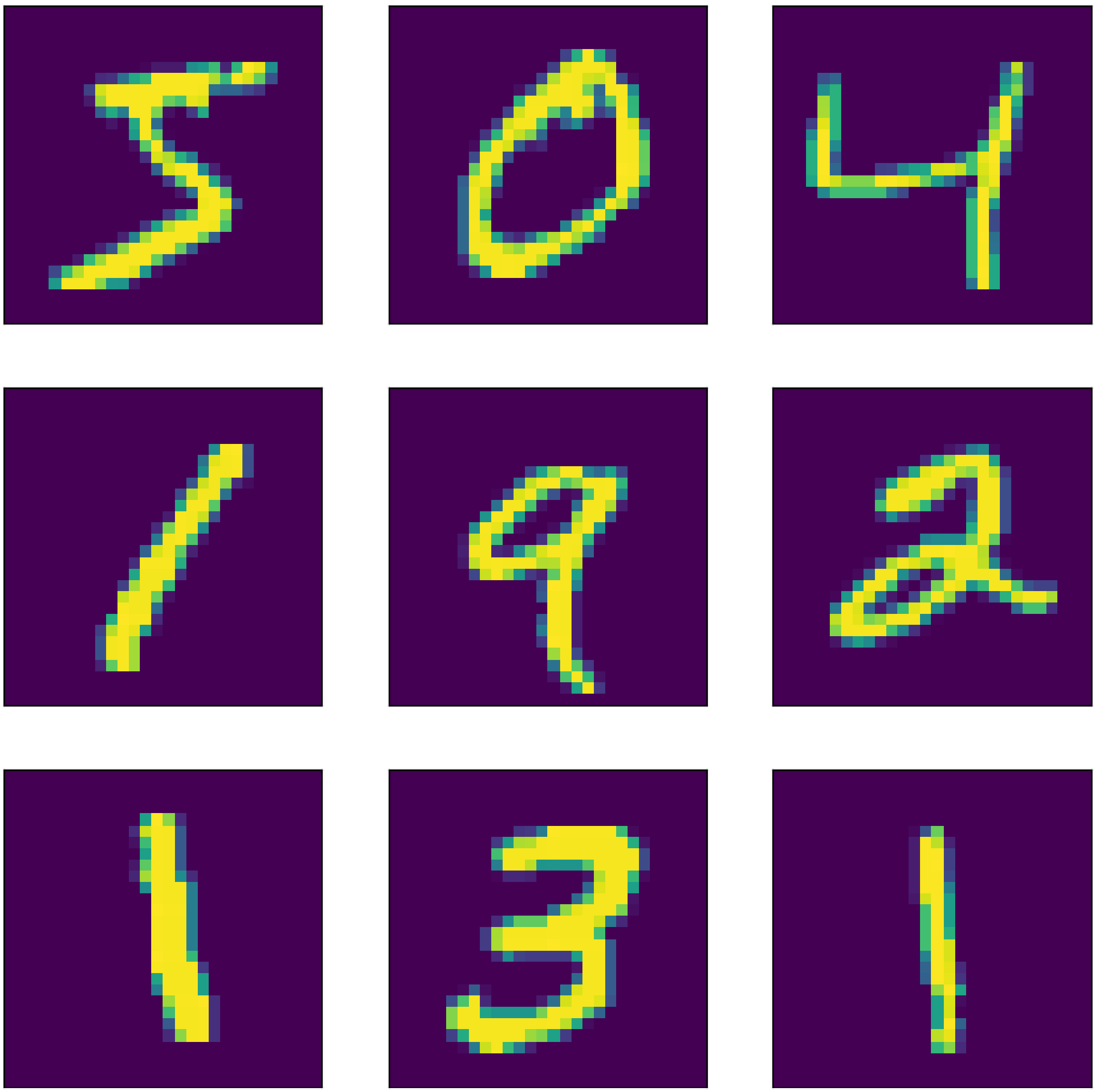
[ ]:
```
(X_train,y_train),(X_test,y_test) = mnist.load_data()
```

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

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

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

[ ]:
```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([])
```



[ ]:
```python
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)
```

[ ]:
```
# 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)
```

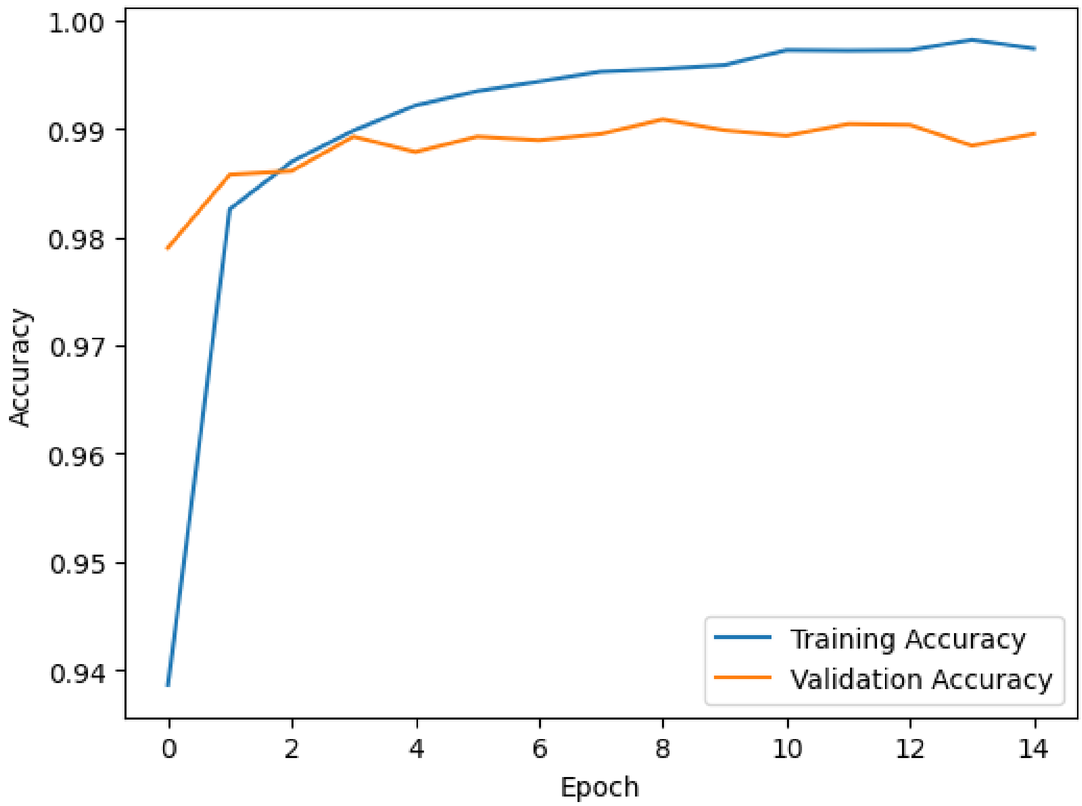
```
Model: "sequential_6"

_________________________________________________________________
 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
```

```
[ ]: # 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
```

```
[ ]: 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 keras.applications import VGG19
from keras import layers,models
from keras.utils import to_categorical
```

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

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/mnist.npz
11490434/11490434 [==============================] - 1s 0us/step
X_train shape: (60000, 28, 28)
```

[ ]:
```
X_train = np.repeat(tf.image.resize(X_train[...,np.newaxis],(32,32)).
 ↪numpy(),3,axis=-1)
X_test=np.repeat(tf.image.resize(X_test[...,np.newaxis],(32,32)).
 ↪numpy(),3,axis=-1)
print(f'X_train shape: {X_train[0].shape}')
```

```
X_train shape: (32, 32, 3)
```

[ ]:
```
X_train = X_train.astype('float32')/255
X_test = X_test.astype('float32')/255

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

[ ]:
```
base_model = VGG19(include_top=False,
    weights='imagenet',
    input_shape=(32,32,3))
base_model.trainable = False

model = models.Sequential([
    base_model,
    layers.Flatten(),
    layers.Dense(256,activation='relu'),
    layers.Dense(10,activation='softmax')])

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

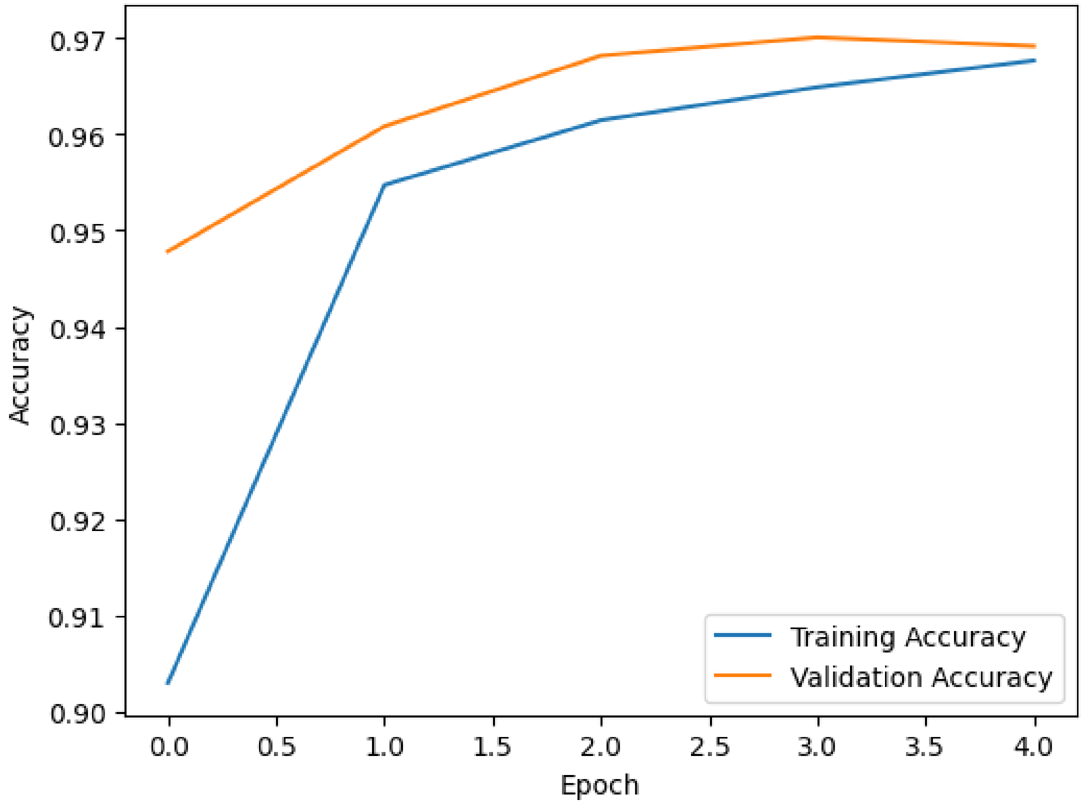
```
history = model.fit(X_train,y_train,
                    epochs=5,batch_size=64,
                    validation_split=0.2)
model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5
80134624/80134624 [==============================] - 3s 0us/step
Epoch 1/5
750/750 [==============================] - 21s 17ms/step - loss: 0.3496 -
accuracy: 0.9030 - val_loss: 0.1688 - val_accuracy: 0.9478
Epoch 2/5
750/750 [==============================] - 12s 16ms/step - loss: 0.1468 -
accuracy: 0.9547 - val_loss: 0.1252 - val_accuracy: 0.9608
Epoch 3/5
750/750 [==============================] - 12s 17ms/step - loss: 0.1208 -
accuracy: 0.9615 - val_loss: 0.1032 - val_accuracy: 0.9682
Epoch 4/5
750/750 [==============================] - 14s 18ms/step - loss: 0.1080 -
accuracy: 0.9649 - val_loss: 0.0971 - val_accuracy: 0.9701
Epoch 5/5
750/750 [==============================] - 12s 16ms/step - loss: 0.0979 -
accuracy: 0.9677 - val_loss: 0.0993 - val_accuracy: 0.9692
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg19 (Functional)          (None, 1, 1, 512)         20024384

 flatten (Flatten)           (None, 512)               0

 dense (Dense)               (None, 256)               131328

 dense_1 (Dense)             (None, 10)                2570

=================================================================
Total params: 20158282 (76.90 MB)
Trainable params: 133898 (523.04 KB)
Non-trainable params: 20024384 (76.39 MB)
_________________________________________________________________
```

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

```
313/313 [==============================] - 4s 12ms/step - loss: 0.1007 -
accuracy: 0.9686
```

[ ]:
```
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();
```



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

[ ]:
```
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
```

[ ]:
```
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
```

[ ]:
```
# 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)
```

[ ]:
```
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)
_________________________________________________________________
```

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

)
```

[ ]:
```
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
```

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

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

[ ]: [0.6362121105194092, 0.7153599858283997]

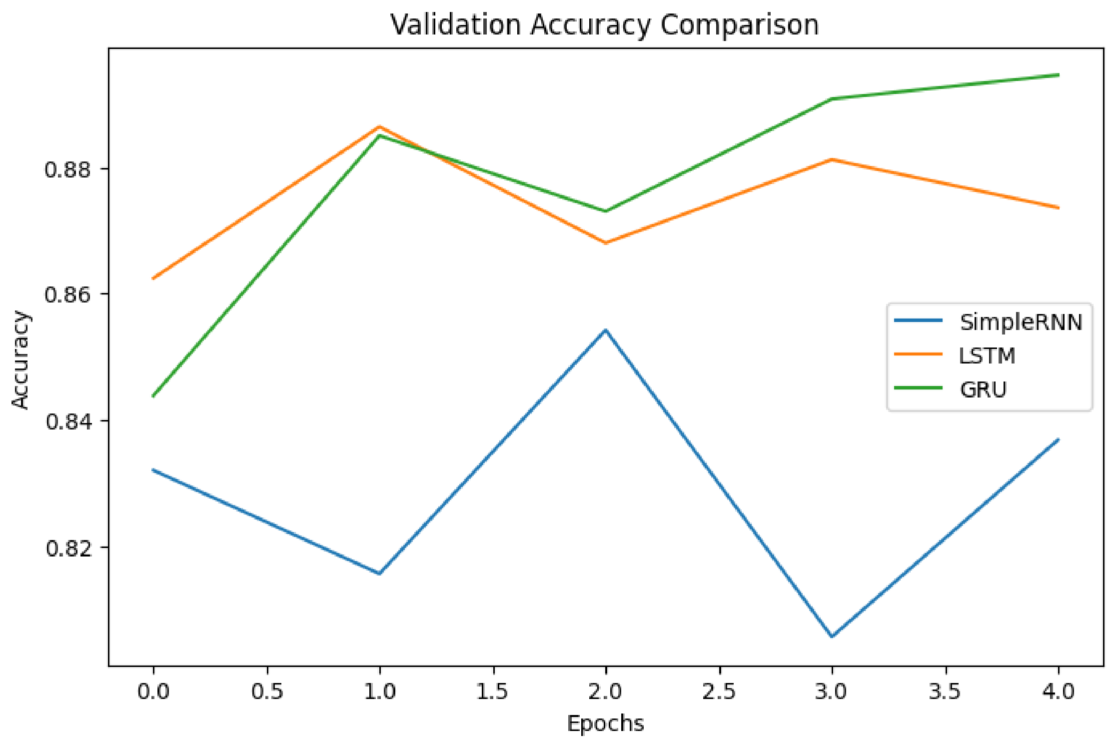
6 8. Analyse and visualize the performance change while using LSTM and GRU instead of simple RNN

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

# Load the IMDB dataset
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':
```

```python
        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): 0.8351200222969055
Test accuracy (LSTM): 0.8726400136947632
Test accuracy (GRU): 0.8887199759483337
```

[ ]:
```python
# Plot validation accuracy
plt.figure(figsize=(8, 5))
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()
```



[ ]:
```python
#predictions
predictions_rnn = model_rnn.predict(test_data)
predictions_lstm = model_lstm.predict(test_data)
predictions_gru = model_gru.predict(test_data)
```

```
782/782 [==============================] - 36s 45ms/step
782/782 [==============================] - 6s 8ms/step
782/782 [==============================] - 5s 6ms/step
```

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

[2]:
```
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-2-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)
```

[2]:

	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

[3]:
```
# Normalize the data
scaler = MinMaxScaler(feature_range=(0, 1))
data_scaled = scaler.fit_transform(data)
data_scaled
```

[3]:
```
array([[0.44754647, 0.42956052, 0.4862573 , 0.4472731 ],
       [0.4472731 , 0.43641819, 0.4826868 , 0.457111454],
       [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]])
```

[4]:
```
# Split the data into training and testing sets
n = int(len(data_scaled) * 0.8)
train_data = data_scaled[:n]
```

```python
test_data = data_scaled[n:]

# Define the parameters
n_input = 3
n_features = 4
```

[5]:
```python
# Create time series generators
generator_train = TimeseriesGenerator(train_data, train_data, length=n_input)
generator_test = TimeseriesGenerator(test_data, test_data, length=n_input)
```

[7]:
```python
generator_train[0]
```

[7]:
```
(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]],

        [[0.4472731 , 0.43641819, 0.4826868 , 0.45711454],
         [0.45711454, 0.43628631, 0.4969688 , 0.43838846],
         [0.43838846, 0.42072467, 0.47093391, 0.42430973]],

        [[0.45711454, 0.43628631, 0.4969688 , 0.43838846],
         [0.43838846, 0.42072467, 0.47093391, 0.42430973],
         [0.42430973, 0.40463552, 0.42868301, 0.37879306]],

        …,

        [[0.31824084, 0.32300287, 0.36515788, 0.31947102],
         [0.31947102, 0.30348488, 0.34804924, 0.29992482],
         [0.31646391, 0.30058356, 0.34447874, 0.29527747]],

        [[0.31947102, 0.30348488, 0.34804924, 0.29992482],
         [0.31646391, 0.30058356, 0.34447874, 0.29527747],
         [0.21367551, 0.28489005, 0.25774538, 0.29158693]],

        [[0.31646391, 0.30058356, 0.34447874, 0.29527747],
         [0.21367551, 0.28489005, 0.25774538, 0.29158693],
         [0.29650765, 0.28159309, 0.33689143, 0.28652952]]]),
 array([[0.43838846, 0.42072467, 0.47093391, 0.42430973],
        [0.42430973, 0.40463552, 0.42868301, 0.37879306],
        [0.37879306, 0.40001978, 0.43746048, 0.41583516],
        [0.41583516, 0.39645907, 0.44251869, 0.39396528],
        [0.39396528, 0.37535854, 0.39163908, 0.34325451],
        [0.34325451, 0.35702746, 0.39878008, 0.3588368 ],
        [0.3588368 , 0.36863275, 0.38821735, 0.3824836 ],
        [0.3824836 , 0.36995153, 0.4200543 , 0.37865637],
        [0.37865637, 0.3665227 , 0.41499609, 0.37305221],
        [0.37305221, 0.35847812, 0.39015137, 0.3629374 ],
```

```
[0.3629374 , 0.3665227 , 0.41454978, 0.38467059],
[0.38467059, 0.38313936, 0.4103842 , 0.37687944],
[0.37687944, 0.40740497, 0.40443337, 0.36854155],
[0.36854155, 0.38762322, 0.41306207, 0.36608119],
[0.36608119, 0.38946952, 0.41707889, 0.37086523],
[0.37086523, 0.40226171, 0.42883178, 0.40517359],
[0.40517359, 0.39856912, 0.45233756, 0.39697239],
[0.39697239, 0.37825987, 0.41157437, 0.35528294],
[0.35528294, 0.35623619, 0.37943988, 0.36799481],
[0.36799481, 0.41663644, 0.41187191, 0.41788546],
[0.41788546, 0.43167057, 0.45947856, 0.44891334],
[0.44891334, 0.44235271, 0.46989251, 0.44781985],
[0.44781985, 0.48152056, 0.509168  , 0.49374658],
[0.49374658, 0.5024892 , 0.52895451, 0.50768863],
[0.50768863, 0.53730507, 0.53133485, 0.54199699],
[0.54199699, 0.5377007 , 0.53892216, 0.54951476],
[0.54951476, 0.59071577, 0.53282255, 0.60090897],
[0.60090897, 0.58267119, 0.61925838, 0.58532668],
[0.51971706, 0.5747585 , 0.59084316, 0.58970066],
[0.49935074, 0.55233919, 0.56867631, 0.53010525],
[0.48158147, 0.5055224 , 0.54636069, 0.50126435],
[0.50126435, 0.52306221, 0.5707591 , 0.52846501],
[0.52846501, 0.51541327, 0.57269312, 0.5234076 ],
[0.5234076 , 0.50024727, 0.55022874, 0.48417851],
[0.48417851, 0.5255679 , 0.54160003, 0.54760115],
[0.54760115, 0.53901949, 0.58905791, 0.55621241],
[0.55621241, 0.5656589 , 0.59649645, 0.56851422],
[0.56851422, 0.56750519, 0.62907725, 0.55539229],
[0.55539229, 0.53110679, 0.59054562, 0.55347868],
[0.55347868, 0.58583627, 0.62491167, 0.605693  ],
[0.605693  , 0.57963799, 0.61345632, 0.5707012 ],
[0.5707012 , 0.5565593 , 0.58890914, 0.58122608],
[0.52723483, 0.56143879, 0.59902555, 0.57234144],
[0.57234144, 0.54745969, 0.53475657, 0.56810416],
[0.56810416, 0.54337147, 0.64350802, 0.56810416],
[0.56810416, 0.61049751, 0.5839997 , 0.62359896],
[0.55498223, 0.57831921, 0.60467884, 0.53393248],
[0.48923592, 0.49022452, 0.49786142, 0.44918671],
[0.39328185, 0.44564967, 0.45323019, 0.43087069],
[0.3372403 , 0.41914213, 0.39223417, 0.39546884],
[0.39546884, 0.44169332, 0.45561052, 0.45684117],
[0.45684117, 0.43602255, 0.45397404, 0.40038956],
[0.40038956, 0.38195246, 0.43180719, 0.37359896],
[0.37359896, 0.40226171, 0.42273218, 0.41173455],
[0.41173455, 0.41426264, 0.43954327, 0.41788546],
[0.41788546, 0.39843724, 0.44965969, 0.40626709],
[0.40626709, 0.38801886, 0.40116041, 0.39601558],
```

```
[0.39601558, 0.39250272, 0.37125749, 0.40667715],
[0.40667715, 0.39461277, 0.42764161, 0.40394341],
[0.40394341, 0.41281197, 0.43493138, 0.43210087],
[0.43210087, 0.41215258, 0.42660022, 0.40175642],
[0.40175642, 0.38604068, 0.40517722, 0.39628896],
[0.39628896, 0.39883288, 0.42332726, 0.41569847],
[0.41569847, 0.39632719, 0.4404359 , 0.40654046],
[0.40654046, 0.41861462, 0.44489902, 0.41938901],
[0.33683024, 0.4096469 , 0.39178785, 0.4236263 ],
[0.4236263 , 0.45039728, 0.43150965, 0.47078321],
[0.44153226, 0.45329861, 0.49801019, 0.47160334],
[0.40804401, 0.43826448, 0.46929743, 0.43770503],
[0.41323811, 0.4509248 , 0.47495072, 0.46094177],
[0.46094177, 0.4399789 , 0.52687172, 0.46094177],
[0.46094177, 0.50776433, 0.52687172, 0.46094177],
[0.46094177, 0.46622268, 0.49057165, 0.45356069],
[0.45356069, 0.43285747, 0.43106334, 0.38070667],
[0.38070667, 0.3938215 , 0.43954327, 0.41091443],
[0.41091443, 0.42956052, 0.46780972, 0.43838846],
[0.43838846, 0.44274834, 0.49369584, 0.44590623],
[0.44590623, 0.4448584 , 0.50619258, 0.45588436],
[0.45588436, 0.47136593, 0.5213672 , 0.49347321],
[0.49347321, 0.48059741, 0.45442035, 0.39984281],
[0.39984281, 0.38102931, 0.39833377, 0.35227583],
[0.35227583, 0.35320299, 0.38613456, 0.36662794],
[0.36662794, 0.35610432, 0.40130918, 0.34872198],
[0.34872198, 0.34080644, 0.3900026 , 0.33601011],
[0.33601011, 0.35267548, 0.39089523, 0.35610306],
[0.35610306, 0.33882826, 0.37854725, 0.33108939],
[0.33108939, 0.33658633, 0.33704021, 0.35090897],
[0.31345681, 0.36098381, 0.36634805, 0.37086523],
[0.32657873, 0.34845538, 0.38063004, 0.34858529],
[0.33081602, 0.35399426, 0.3803325 , 0.36252734],
[0.35733324, 0.35188421, 0.40993789, 0.35692318],
[0.33888054, 0.34924665, 0.39401941, 0.36471432],
[0.3454415 , 0.34410339, 0.40116041, 0.35350601],
[0.33518999, 0.34898289, 0.3900026 , 0.35733324],
[0.35077228, 0.36612706, 0.40696247, 0.36881493],
[0.36881493, 0.35175233, 0.40458214, 0.34858529],
[0.34858529, 0.35359863, 0.40130918, 0.37045517],
[0.37045517, 0.35267548, 0.40011902, 0.35746993],
[0.35746993, 0.34014704, 0.38866367, 0.35118234],
[0.35118234, 0.35425802, 0.39639975, 0.36813149],
[0.36813149, 0.35175233, 0.41008666, 0.36430426],
[0.36430426, 0.35847812, 0.40770633, 0.36348414],
[0.35104565, 0.36401701, 0.39892885, 0.37660607],
[0.37660607, 0.36507204, 0.35072712, 0.37305221],
```

```
        [0.37305221, 0.383535  , 0.41603749, 0.39218835],
        [0.39218835, 0.3744354 , 0.33971808, 0.39082149],
        [0.36717469, 0.39303023, 0.41068174, 0.38877119],
        [0.33231957, 0.36850087, 0.38687842, 0.34858529],
        [0.32575861, 0.35531305, 0.37973742, 0.35938354],
        [0.33327638, 0.40991065, 0.38791981, 0.42143931],
        [0.35432613, 0.39936039, 0.41083051, 0.38672089],
        [0.3422977 , 0.38881013, 0.39773868, 0.39779251],
        [0.36922499, 0.36995153, 0.42451742, 0.37182203],
        [0.34079415, 0.35887376, 0.39610221, 0.35022553],
        [0.32521186, 0.33183871, 0.37914234, 0.34257108],
        [0.34257108, 0.33632257, 0.38940752, 0.33806042],
        [0.33806042, 0.32142033, 0.37899357, 0.32671542],
        [0.32671542, 0.31548581, 0.36872838, 0.31646391],
        [0.31646391, 0.30928753, 0.35251237, 0.31509705],
        [0.31509705, 0.3191784 , 0.36396772, 0.32493849],
        [0.29063013, 0.32906927, 0.34150333, 0.33272963],
        [0.33272963, 0.32669546, 0.33346971, 0.33190951],
        [0.31824084, 0.32300287, 0.36515788, 0.31947102],
        [0.31947102, 0.30348488, 0.34804924, 0.29992482],
        [0.31646391, 0.30058356, 0.34447874, 0.29527747],
        [0.21367551, 0.28489005, 0.25774538, 0.29158693],
        [0.29650765, 0.28159309, 0.33689143, 0.28652952],
        [0.27395435, 0.27513105, 0.32335329, 0.28584609]]))
```

[12]:

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


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

```
Epoch 1/50
18/18 [==============================] - 3s 11ms/step - loss: 0.0184 - accuracy:
0.5425
Epoch 2/50
18/18 [==============================] - 0s 9ms/step - loss: 0.0106 - accuracy:
0.8210
Epoch 3/50
18/18 [==============================] - 0s 11ms/step - loss: 0.0054 - accuracy:
0.8210
Epoch 4/50
```

```
18/18 [==============================] - 0s 10ms/step - loss: 0.0038 - accuracy:
0.8210
Epoch 5/50
18/18 [==============================] - 0s 11ms/step - loss: 0.0030 - accuracy:
0.8024
Epoch 6/50
18/18 [==============================] - 0s 10ms/step - loss: 0.0023 - accuracy:
0.6370
Epoch 7/50
18/18 [==============================] - 0s 11ms/step - loss: 0.0019 - accuracy:
0.5416
Epoch 8/50
18/18 [==============================] - 0s 11ms/step - loss: 0.0013 - accuracy:
0.5235
Epoch 9/50
18/18 [==============================] - 0s 10ms/step - loss: 8.8252e-04 -
accuracy: 0.4914
Epoch 10/50
18/18 [==============================] - 0s 10ms/step - loss: 6.2975e-04 -
accuracy: 0.5533
Epoch 11/50
18/18 [==============================] - 0s 10ms/step - loss: 4.9325e-04 -
accuracy: 0.7147
Epoch 12/50
18/18 [==============================] - 0s 9ms/step - loss: 4.4667e-04 -
accuracy: 0.8210
Epoch 13/50
18/18 [==============================] - 0s 11ms/step - loss: 4.3118e-04 -
accuracy: 0.8210
Epoch 14/50
18/18 [==============================] - 0s 10ms/step - loss: 4.2752e-04 -
accuracy: 0.8210
Epoch 15/50
18/18 [==============================] - 0s 11ms/step - loss: 4.2627e-04 -
accuracy: 0.8210
Epoch 16/50
18/18 [==============================] - 0s 7ms/step - loss: 4.1982e-04 -
accuracy: 0.8210
Epoch 17/50
18/18 [==============================] - 0s 6ms/step - loss: 4.1685e-04 -
accuracy: 0.8210
Epoch 18/50
18/18 [==============================] - 0s 6ms/step - loss: 4.1475e-04 -
accuracy: 0.8210
Epoch 19/50
18/18 [==============================] - 0s 6ms/step - loss: 4.1284e-04 -
accuracy: 0.8210
Epoch 20/50
```

```
18/18 [==============================] - 0s 7ms/step - loss: 4.1058e-04 -
accuracy: 0.8210
Epoch 21/50
18/18 [==============================] - 0s 6ms/step - loss: 4.0991e-04 -
accuracy: 0.8210
Epoch 22/50
18/18 [==============================] - 0s 6ms/step - loss: 4.0822e-04 -
accuracy: 0.8210
Epoch 23/50
18/18 [==============================] - 0s 6ms/step - loss: 4.0643e-04 -
accuracy: 0.8210
Epoch 24/50
18/18 [==============================] - 0s 7ms/step - loss: 4.0544e-04 -
accuracy: 0.8210
Epoch 25/50
18/18 [==============================] - 0s 6ms/step - loss: 4.0428e-04 -
accuracy: 0.8210
Epoch 26/50
18/18 [==============================] - 0s 6ms/step - loss: 4.0155e-04 -
accuracy: 0.8210
Epoch 27/50
18/18 [==============================] - 0s 7ms/step - loss: 3.9923e-04 -
accuracy: 0.8210
Epoch 28/50
18/18 [==============================] - 0s 6ms/step - loss: 3.9960e-04 -
accuracy: 0.8210
Epoch 29/50
18/18 [==============================] - 0s 6ms/step - loss: 3.9584e-04 -
accuracy: 0.8210
Epoch 30/50
18/18 [==============================] - 0s 7ms/step - loss: 3.9285e-04 -
accuracy: 0.8210
Epoch 31/50
18/18 [==============================] - 0s 9ms/step - loss: 3.9049e-04 -
accuracy: 0.8210
Epoch 32/50
18/18 [==============================] - 0s 17ms/step - loss: 3.8848e-04 -
accuracy: 0.8210
Epoch 33/50
18/18 [==============================] - 0s 12ms/step - loss: 3.8625e-04 -
accuracy: 0.8210
Epoch 34/50
18/18 [==============================] - 0s 6ms/step - loss: 3.8286e-04 -
accuracy: 0.8210
Epoch 35/50
18/18 [==============================] - 0s 6ms/step - loss: 3.8076e-04 -
accuracy: 0.8210
Epoch 36/50
```

```
18/18 [==============================] - 0s 7ms/step - loss: 3.7866e-04 -
accuracy: 0.8210
Epoch 37/50
18/18 [==============================] - 0s 6ms/step - loss: 3.7710e-04 -
accuracy: 0.8210
Epoch 38/50
18/18 [==============================] - 0s 6ms/step - loss: 3.7157e-04 -
accuracy: 0.8210
Epoch 39/50
18/18 [==============================] - 0s 7ms/step - loss: 3.6905e-04 -
accuracy: 0.8210
Epoch 40/50
18/18 [==============================] - 0s 7ms/step - loss: 3.6683e-04 -
accuracy: 0.8210
Epoch 41/50
18/18 [==============================] - 0s 6ms/step - loss: 3.6294e-04 -
accuracy: 0.8210
Epoch 42/50
18/18 [==============================] - 0s 7ms/step - loss: 3.5956e-04 -
accuracy: 0.8210
Epoch 43/50
18/18 [==============================] - 0s 6ms/step - loss: 3.5769e-04 -
accuracy: 0.8210
Epoch 44/50
18/18 [==============================] - 0s 6ms/step - loss: 3.5297e-04 -
accuracy: 0.8210
Epoch 45/50
18/18 [==============================] - 0s 6ms/step - loss: 3.4994e-04 -
accuracy: 0.8210
Epoch 46/50
18/18 [==============================] - 0s 6ms/step - loss: 3.4705e-04 -
accuracy: 0.8210
Epoch 47/50
18/18 [==============================] - 0s 6ms/step - loss: 3.4073e-04 -
accuracy: 0.8210
Epoch 48/50
18/18 [==============================] - 0s 6ms/step - loss: 3.3766e-04 -
accuracy: 0.8210
Epoch 49/50
18/18 [==============================] - 0s 6ms/step - loss: 3.3597e-04 -
accuracy: 0.8210
Epoch 50/50
18/18 [==============================] - 0s 7ms/step - loss: 3.2919e-04 -
accuracy: 0.8210
```

```
[12]: <keras.src.callbacks.History at 0x7866e93897b0>
```

[13]:
```
print(model.summary())
```

```
Model: "sequential_3"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_3 (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
```

[15]:
```
# Evaluate the model on the test set
model.evaluate(generator_test)
```

```
5/5 [==============================] - 0s 10ms/step - loss: 8.8715e-04 -
accuracy: 0.5572
```

[15]:
```
[0.0008871516329236329, 0.5571687817573547]
```

[16]:
```
# Make predictions on the test set
predictions = model.predict(generator_test)
predictions
```

```
5/5 [==============================] - 0s 5ms/step
```

[16]:
```
array([[0.11395665, 0.11410186, 0.1309043 , 0.11727935],
       [0.11760904, 0.12378584, 0.13550733, 0.12555383],
       [0.11276945, 0.11939283, 0.12960596, 0.11811133],
       …,
       [0.3904209 , 0.39378923, 0.42502874, 0.3987396 ],
       [0.41211098, 0.42068988, 0.45026392, 0.41997573],
       [0.40956986, 0.42264092, 0.4428761 , 0.41472122]], dtype=float32)
```

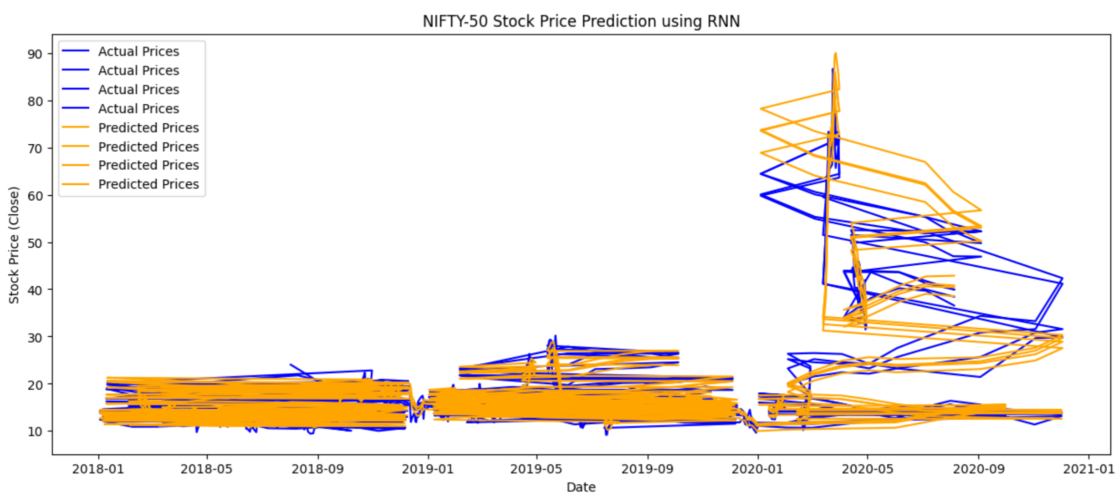
[27]:
```
# Inverse transform the predictions and actual values to the original scale
predictions_original = scaler.inverse_transform(predictions)
test_data_original = scaler.inverse_transform(test_data[n_input:])
test_data_original,predictions_original
```

[27]:
```
(array([[19.465 , 24.035 , 15.2625, 17.7725],
        [17.7725, 20.4225, 15.15  , 19.23  ],
        [19.23  , 19.23  , 16.5675, 17.8825],
        …,
```

```
[43.6075, 43.63  , 40.1975, 41.2475],
       [41.2475, 41.715 , 39.6575, 39.93  ],
       [39.93  , 39.93  , 36.5475, 38.4075]]),
 array([[18.784569, 19.45956 , 17.55406 , 19.027657],
        [19.051777, 20.19387 , 17.863464, 19.633018],
        [18.697712, 19.86076 , 17.46679 , 19.088524],
        …,
        [39.010693, 40.667553, 37.32437 , 39.61929 ],
        [40.597538, 42.707363, 39.02062 , 41.172924],
        [40.41163 , 42.855305, 38.524025, 40.788506]], dtype=float32))
```

[28]:

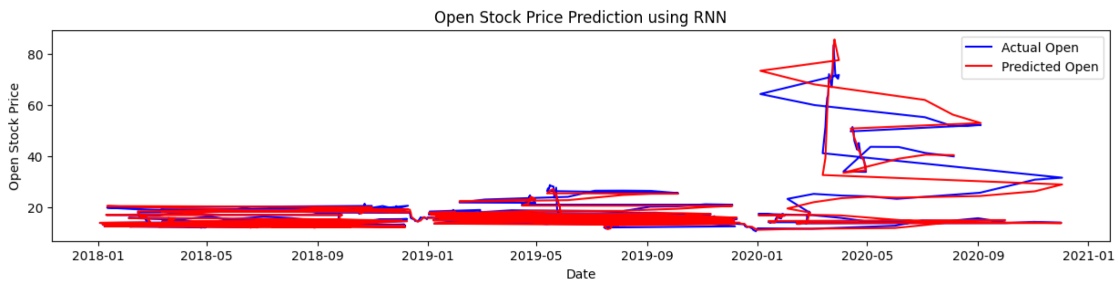
```
# Plot the results
plt.figure(figsize=(15, 6))
plt.plot(data.index[n + n_input:], test_data_original, label='Actual Prices',␣
 ↪color='blue')
plt.plot(data.index[n + n_input:], predictions_original, label='Predicted␣
 ↪Prices', color='orange')
plt.title('NIFTY-50 Stock Price Prediction using RNN')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

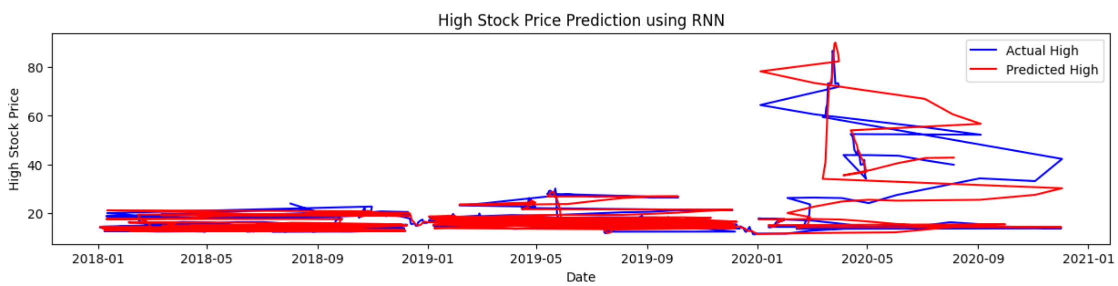


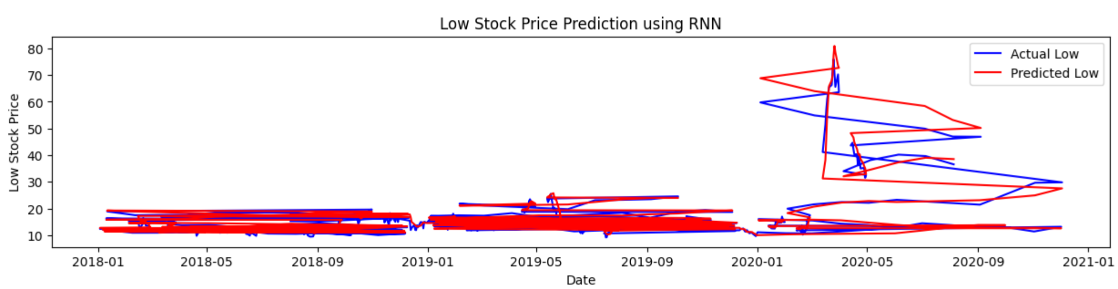
[31]:

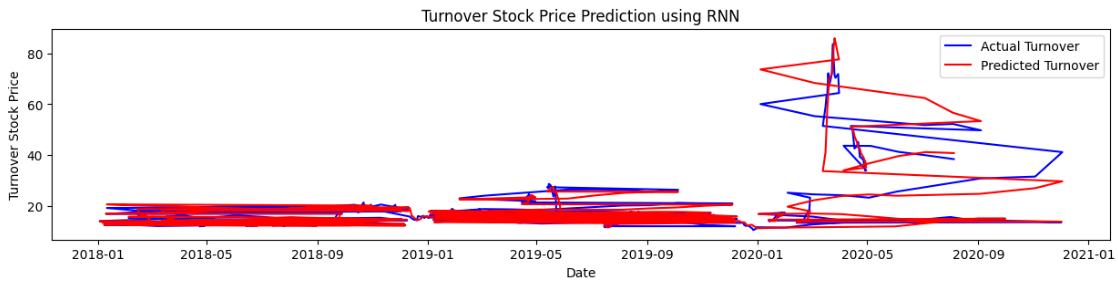
```
# Plot the results for each variable
variables = ['Open', 'High', 'Low', 'Turnover']

for i, variable in enumerate(variables):
    plt.figure(figsize=(15, 3))  # Adjust the height as needed

    # Plot actual prices (test_data_original)
```

```
    plt.plot(data.index[n + n_input:], test_data_original[:, i], label=f'Actual {variable}', color='blue')

    # Plot predicted prices (predictions_original)
    plt.plot(data.index[n + n_input:], predictions_original[:, i], label=f'Predicted {variable}', color='red')

    # Add title and axis labels
    plt.title(f'{variable} Stock Price Prediction using RNN')
    plt.xlabel('Date')
    plt.ylabel(f'{variable} Stock Price')
    plt.legend()
    plt.show()
```








Turnover Stock Price Prediction using RNN
Actual Turnover
Predicted Turnover
Turnover Stock Price
80
60
40
20
2018-01
2018-05
2018-09
2019-01
2019-05
2019-09
2020-01
2020-05
2020-09
2021-01
Date

[ ]: