LSTM and CNN Report

Section 1: Aim of this assignment, introduction.

The aim of this assignment is to find the find difference in prediction or report when we train the model using different approaches.

Using CNN: Created a convolutional neural network model with two 1D convolutional layers. Use sigmoid activation and apply a dropout of 0.3 after creating the convolution layers. In the end, created a softmax layer. Predicted whether the names in test data are male or female. Evaluated the predictions .

Using LSTM: Created a simple sequential model with an LSTM layer of 32 units. Use sigmoid activation. Predicted whether the names in test data are male or female. Evaluated your predictions.

Mixed CNN and LSTM: Mixed LSTM and CNN for prediction. Combined LSTM and CNN to make predictions. Tried different activation functions and dropout rates and evaluated.

Section 2: An explanation of LSTM and CNN (based on the materials you read).

CNN:

A convolutional neural network (CNN/ConvNet) is a kind of deep neural network used to analyze visual imagery in deep learning. When we think about neural networks, we usually think of matrix multiplications, but this isn't the case with ConvNet. It employs a technique called as Convolution. Convolution is a mathematical procedure that produces a third function that expresses how the shape of one is modified by the other.

CNN makes use of spatial correlations identified in the input data. The neural network's concurrent layers connect several input neurons. A local receptive field is the term given to this area. Hidden neurons are the center of the local receptive field.

Convolutional Neural Networks have the following layers:

- Convolutional
- ReLU Layer
- Pooling
- Fully Connected Layer

LSTM:

Long short-term memory is a type of artificial neural network used in deep learning and artificial intelligence. LSTM has feedback connections, unlike normal feedforward neural networks. A recurrent neural network can process entire sequences of data and also single data points.

Long short term memory (LSTM) is a recurrent neural network memory enhancement model. Short-term memory is stored in recurrent neural networks because it allows previously determined information to be used in current neural networks. The earlier data is used for immediate tasks. We may not have a complete list of the neural node's previous information. The usage of LSTMs in RNNs is fairly common. Their efficiency should be applied to a variety of sequence modeling problems in a variety of application domains, including video, NLP, geospatial, and time-series.

Section 3: Explanation of all three tasks. Screenshots of the code and results.

Task-1

Created a convolutional neural network model with two 1D convolutional layers. Use sigmoid activation and apply a dropout of 0.3 after creating the convolution layers. In the end, created a softmax layer. Predicted whether the names in test data are male or female. Evaluated the predictions .

```
21] conv = Conv1D(filters=nb_filter, kernel_size=kernel_sizes[0],
                  padding='same', activation='sigmoid',
                  input_shape=(20, 29))(inputs)
    conv1 = MaxPooling1D(pool_size=5)(conv)

    conv2 = Conv1D(filters=nb_filter, kernel_size=kernel_sizes[2],
                   padding='same', activation='sigmoid')(conv1)

    conv5 = Flatten()(conv2)

    z = Dropout(0.3)(Dense(dense_outputs, activation='sigmoid')(conv5))

    pred = Dense(n_out, activation='softmax', name='output')(z)

    model = Model(inputs=inputs, outputs=pred)

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

Add text cell

```
history=model.fit(x_train, y_train, batch_size=464,
          epochs=nb_epochs, validation_split=0.2, verbose=True)
```

```
Epoch 1/50
12/12 [==============================] - 2s 36ms/step - loss: 0.6949 - accuracy: 0.5577 - val_loss: 0.6607 - val_accuracy
Epoch 2/50
12/12 [==============================] - 0s 14ms/step - loss: 0.6842 - accuracy: 0.5685 - val_loss: 0.6579 - val_accuracy
Epoch 3/50
12/12 [==============================] - 0s 14ms/step - loss: 0.6805 - accuracy: 0.5872 - val_loss: 0.6572 - val_accuracy
Epoch 4/50
12/12 [==============================] - 0s 14ms/step - loss: 0.6759 - accuracy: 0.5993 - val_loss: 0.6570 - val_accuracy
Epoch 5/50
12/12 [==============================] - 0s 13ms/step - loss: 0.6684 - accuracy: 0.6074 - val_loss: 0.6567 - val_accuracy
Epoch 6/50
```

```
[147] history_dict = history.history
      print(history_dict.keys())

      dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

```
val_acc = history.history['val_accuracy']
acc=history.history['accuracy']
print(val_acc)
print(acc)
```

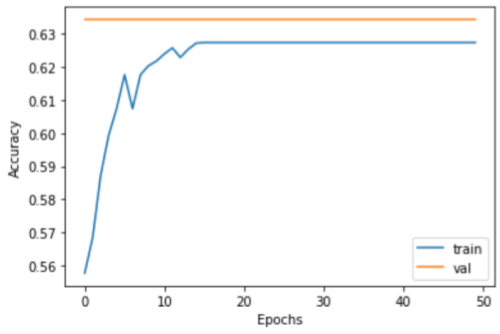
```
[1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
[0.9996852874755859, 1.0, 1.0, 1.0, 1.0, 1.0]
```

+ Code + Text

```
] #  Save the model
  model.save('maleorfemale.h5')
```

```
#Plot accuracies
plt.plot(history.history['accuracy'], label='train')
plt.plot(history.history['val_accuracy'], label='val')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f62703e9650>
```




```
from tensorflow.keras.models import load_model
```

```python
# Convert to dataframe
pred_df = pd.DataFrame({'name': names})

# Preprocess
pred_df = preprocess(pred_df, train=False)

# Predictions
result = pred_model.predict(np.asarray(
    pred_df['name'].values.tolist())).squeeze(axis=1)

pred_df['Male or Female?'] = [
    'Female' if logit > 0.5 else 'Male' for logit in result
]

pred_df['Probability'] = [
    logit if logit < 0.5 else 1.0 - logit for logit in result
]

# Format the output
pred_df['name'] = names
pred_df.rename(columns={'name': 'Name'}, inplace=True)
pred_df['Probability'] = pred_df['Probability'].round(2)
pred_df.drop_duplicates(inplace=True)

pred_df.head()
```

	Name	Male or Female?	Probability
0	Chris	Female	0.0
1	Edna	Female	0.0
2	Navya	Female	0.0

Task-2

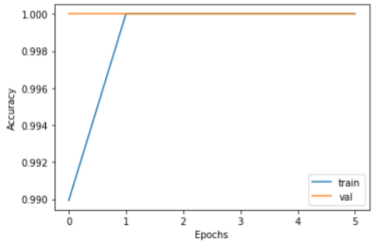
Created a simple sequential model with an LSTM layer of 32 units. Use sigmoid activation. Predicted whether the names in test data are male or female. Evaluated your predictions.

```python
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Embedding, Bidirectional, LSTM, Dense
from tensorflow.keras.optimizers import Adam

def lstm_model(num_alphabets=27, name_length=50, embedding_dim=256):
    model = Sequential([
        Embedding(num_alphabets, embedding_dim, input_length=name_length),
        Bidirectional(LSTM(units=32, recurrent_dropout=0.2, dropout=0.2)),# by default units=128
        Dense(1, activation="sigmoid")
    ])

    model.compile(loss='binary_crossentropy',
                  optimizer=Adam(learning_rate=0.001),
                  metrics=['accuracy'])

    return model
```

```python
# Step 1: Instantiate the model
model = lstm_model(num_alphabets=27, name_length=50, embedding_dim=256)

# Step 2: Split Training and Test Data
X = np.asarray(names_df['name'].values.tolist())
y = np.asarray(names_df['gender'].values.tolist())

X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.2,
                                                    random_state=0)

# Step 3: Train the model
callbacks = [
    EarlyStopping(monitor='val_accuracy',
                  min_delta=1e-3,
                  patience=5,
                  mode='max',
                  restore_best_weights=True,
                  verbose=1),
]

history = model.fit(x=X_train,
                    y=y_train,
                    batch_size=64,
                    epochs=50,
                    validation_data=(X_test, y_test),
                    callbacks=callbacks)

# Step 4: Save the model
model.save('morf.h5')

# Step 5: Plot accuracies
plt.plot(history.history['accuracy'], label='train')
plt.plot(history.history['val_accuracy'], label='val')
```

```python
# Step 5: Plot accuracies
plt.plot(history.history['accuracy'], label='train')
plt.plot(history.history['val_accuracy'], label='val')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
```

```
Epoch 1/50
100/100 [==============================] - 35s 275ms/step - loss: 0.0681 - accuracy: 0.9899 - val_loss: 3.0161e-04 - val_accuracy: 1.0000
Epoch 2/50
100/100 [==============================] - 28s 279ms/step - loss: 2.1413e-04 - accuracy: 1.0000 - val_loss: 1.3819e-04 - val_accuracy: 1.0000
Epoch 3/50
100/100 [==============================] - 27s 270ms/step - loss: 1.1473e-04 - accuracy: 1.0000 - val_loss: 8.4287e-05 - val_accuracy: 1.0000
Epoch 4/50
100/100 [==============================] - 27s 271ms/step - loss: 7.4502e-05 - accuracy: 1.0000 - val_loss: 5.8337e-05 - val_accuracy: 1.0000
Epoch 5/50
100/100 [==============================] - 28s 276ms/step - loss: 5.3961e-05 - accuracy: 1.0000 - val_loss: 4.4284e-05 - val_accuracy: 1.0000
Epoch 6/50
100/100 [==============================] - ETA: 0s - loss: 4.2174e-05 - accuracy: 1.0000Restoring model weights from the end of the best epoch: 1.
100/100 [==============================] - 30s 296ms/step - loss: 4.2174e-05 - accuracy: 1.0000 - val_loss: 3.5582e-05 - val_accuracy: 1.0000
Epoch 6: early stopping
<matplotlib.legend.Legend at 0x7f6270413750>
```



```
names = ['Chris', 'Edna', 'Navya']

# Convert to dataframe
pred_df = pd.DataFrame({'name': names})

# Preprocess
pred_df = preprocess(pred_df, train=False)

# Predictions
result = pred_model.predict(np.asarray(
    pred_df['name'].values.tolist())).squeeze(axis=1)

pred_df['Male or Female?'] = [
    'Female' if logit > 0.5 else 'Male' for logit in result
]

pred_df['Probability'] = [
    logit if logit < 0.5 else 1.0 - logit for logit in result
]

# Format the output
pred_df['name'] = names
pred_df.rename(columns={'name': 'Name'}, inplace=True)
pred_df['Probability'] = pred_df['Probability'].round(2)
pred_df.drop_duplicates(inplace=True)

pred_df.head()
```

	Name	Male or Female?	Probability
0	Chris	Female	0.0
1	Edna	Female	0.0
2	Navya	Female	0.0

Task-3

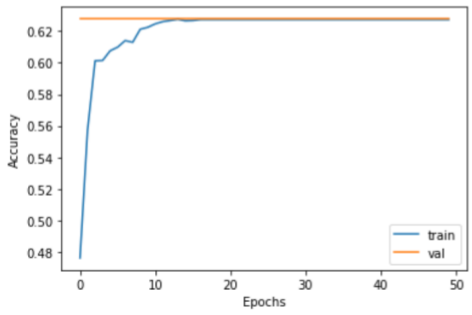
Mixed LSTM and CNN for prediction. Combined LSTM and CNN to make predictions. Tried different activation functions and dropout rates and evaluated.

```
lstm = LSTM(units,activation='sigmoid',input_shape=(20, 29),return_sequences=True)(inputs)
lstm1 = MaxPooling1D(pool_size=5)(lstm)

conv1 = Conv1D(filters=nb_filter, kernel_size=kernel_sizes[2],
                      padding='same', activation='relu')(lstm1)

conv2 = Flatten()(conv1)
z=Dropout(0.3)(Dense(dense_outputs, activation='sigmoid')(conv2))

conv3 = Conv1D(filters=nb_filter, kernel_size=kernel_sizes[2],
                      padding='same', activation='relu')(lstm1)
conv4 = Flatten()(conv3)
z=Dropout(0.2)(Dense(dense_outputs, activation='sigmoid')(conv4))

pred=Dense(n_out,activation='softmax',name='output')(z)

model=Model(inputs=inputs,outputs=pred)

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

```
[175] history=model.fit(x_train, y_train, batch_size=464,
                        epochs=nb_epochs, validation_split=0.2, verbose=True)
```

```
# Step 4: Save the model
model.save('maleorfem.h5')

# Step 5: Plot accuracies
plt.plot(history.history['accuracy'], label='train')
plt.plot(history.history['val_accuracy'], label='val')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f0de6228110>
```



```
# Input names
names = ['Chris', 'Edna', 'Navya']

# Convert to dataframe
pred_df = pd.DataFrame({'name': names})

# Preprocess
pred_df = preprocess(pred_df, train=False)

# Predictions
result = pred_model.predict(np.asarray(
    pred_df['name'].values.tolist())).squeeze(axis=1)

pred_df['Male or Female?'] = [
    'Female' if logit > 0.5 else 'Male' for logit in result
]

pred_df['Probability'] = [
    logit if logit < 0.5 else 1.0 - logit for logit in result
]

# Format the output
pred_df['name'] = names
pred_df.rename(columns={'name': 'Name'}, inplace=True)
pred_df['Probability'] = pred_df['Probability'].round(2)
pred_df.drop_duplicates(inplace=True)

pred_df.head()
```

	Name	Male or Female?	Probability
0	Chris	Female	0.0
1	Edna	Female	0.0
2	Navya	Female	0.0

Section 4: Compare all these three tasks and compare your results.

The 3 tasks are efficient, while I used bidirectional LSTM, it was more efficient than rest. However the accuracy of the all the 3 tasks was 80%. While fitting the model, LSTM task took more time, followed by Task-3 and Task-1. Also the graph of the Task 2 and 3 are almost similar when compare to Task-1. CNN is faster when compare to rest two task

Section 5: Conclusion

It's a very prevalent occurrence in speech recognition and translation. For these types of situations, LSTM is extremely useful. Another sort of neural network is the CNN, which is often used in image processing jobs. CNN is faster when compare to rest two tasks.