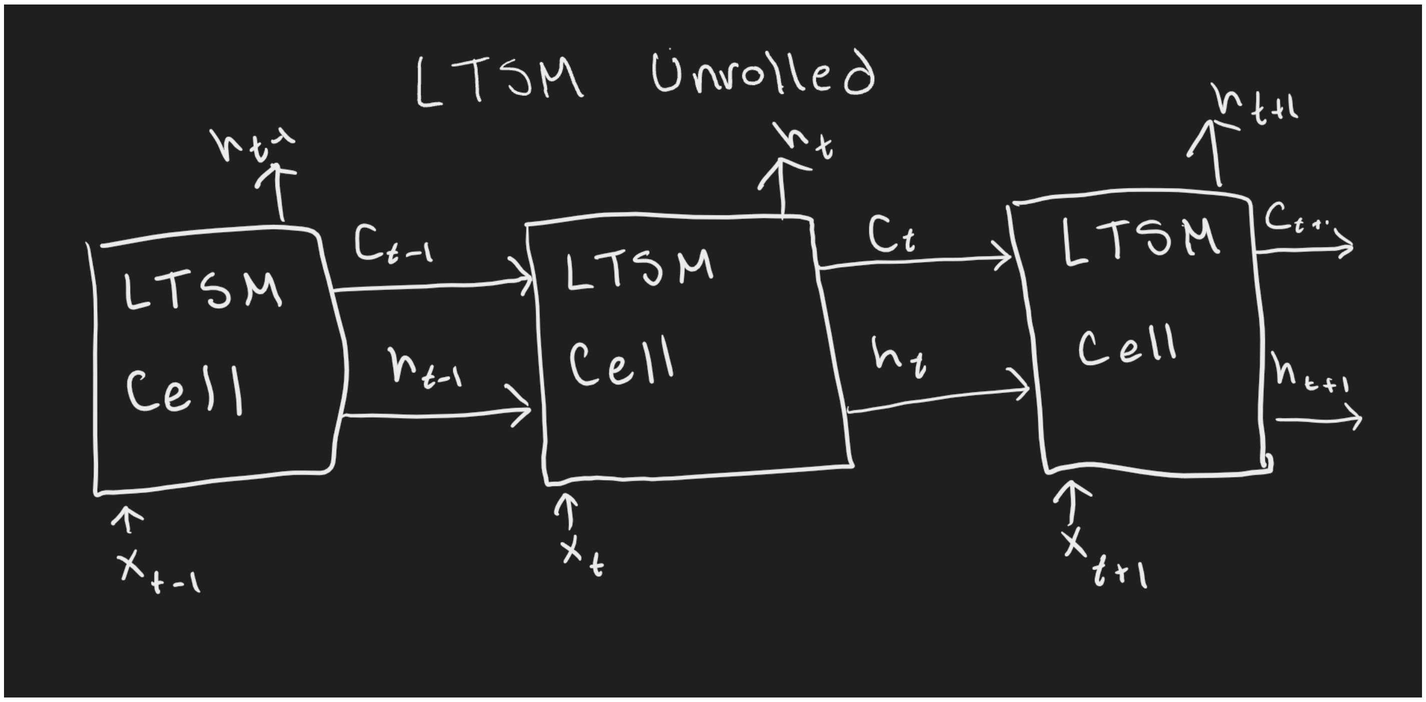
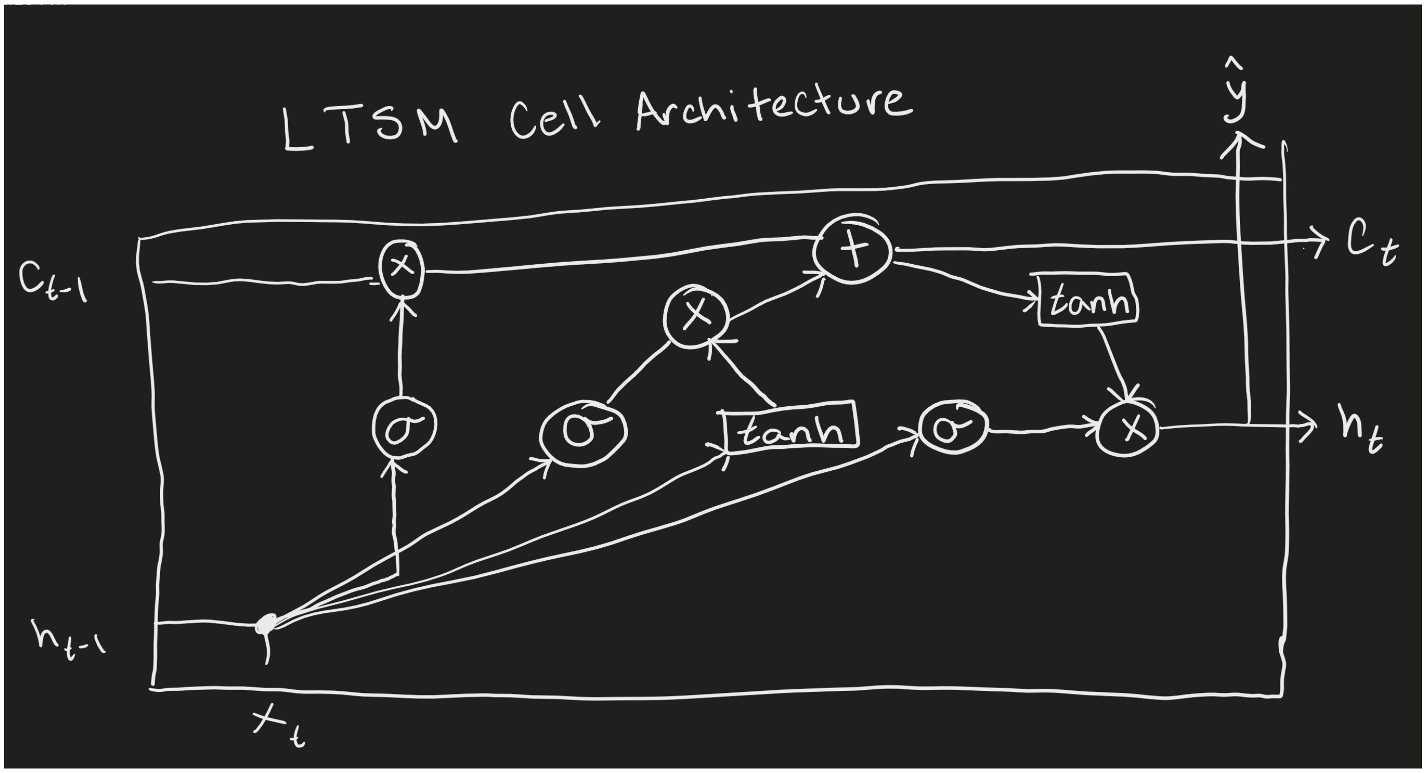
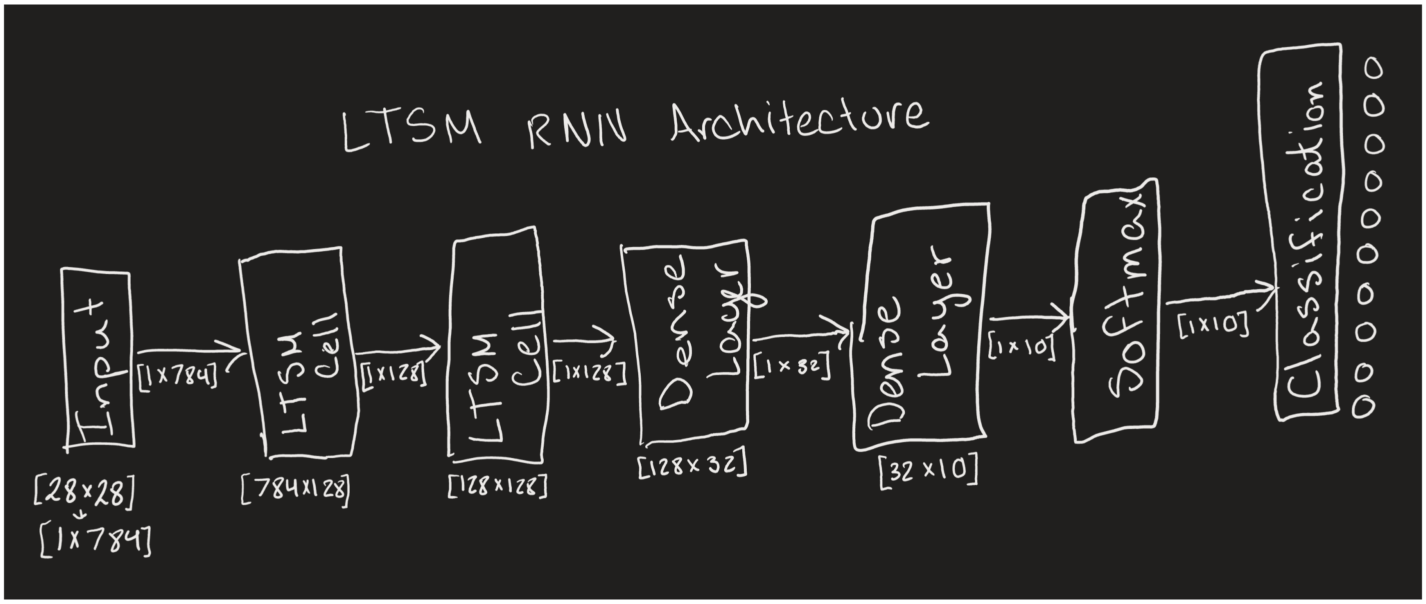

LTSM Unrolled
LTSM Cell
LTSM Cell
LTSM Cell
h_{t-1}
h_t
h_{t+1}
C_{t-1}
h_{t-1}
C_t
h_t
C_{t+1}
h_{t+1}
x_{t-1}
x_t
x_{t+1}


LTSM Cell Architecture
$\hat{y}$
C_{t-1}
C_t
h_{t-1}
h_t
x_t
×
+
×
×
σ
σ
σ
tanh
tanh



While CNNs have classically been used to classify images, it is also possible to do image classification using RNN. The model I used takes input that is of dimensions [28x28]. This can be flattened to [1x784]. The first 2 LTSM cells output a [1x128] vector, the dense layers shorten the vector to [1x32] and [1x10]. The reason the last layer has 10 output cells is because there are 10 classes that need to be classified. By performing a softmax on the last layer, I generate a set of probabilities for each of the classes.

```
import os
import tensorflow as tf
import tensorflow_datasets
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM, Conv2D, MaxPooling2D, Fla

```

Load Data

Divide by 255 to regularize data

```
fashion_mnist = tf.keras.datasets.fashion_mnist
```

```
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
x_train = x_train / 255.0
x_test = x_test / 255.0
```

Create the RNN model

```
RNN_model = Sequential([
    LSTM(128, input_shape=(x_train.shape[1:]), activation='relu', return_sequences=
    LSTM(128, activation='relu'),
    Dense(32, activation='relu'),
    Dense(10, activation='softmax')
])

RNN_model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics
```

```
RNN_model.fit(x_train, y_train, epochs=5)
```

```
Epoch 1/5
1875/1875 [==============================] - 126s 66ms/step - loss: 0.6642 - acc
Epoch 2/5
1875/1875 [==============================] - 125s 66ms/step - loss: 0.4070 - acc
Epoch 3/5
1875/1875 [==============================] - 124s 66ms/step - loss: 0.3562 - acc
Epoch 4/5
1875/1875 [==============================] - 125s 66ms/step - loss: 0.3262 - acc
Epoch 5/5
1875/1875 [==============================] - 125s 67ms/step - loss: 0.3044 - acc
<keras.callbacks.History at 0x7f8639e72d10>
```

```
y_pred = RNN_model.predict(x_test)
```

```
from sklearn.metrics import confusion_matrix, f1_score
import numpy as np
y_pred = np.argmax(y_pred, axis=1)
print(confusion_matrix(y_test, y_pred))
f1_score(y_test, y_pred, average='macro')
```

```
[[834   2  14  62   1   1  80   0   6   0]
 [  1 968   3  22   1   0   4   0   1   0]
 [ 11   2 790  14 133   0  50   0   0   0]
 [ 16  13   9 922  18   0  19   0   3   0]
 [  0   1  78  51 835   1  32   0   2   0]
 [  0   0   0   1   0 946   0  30   2  21]
 [151   3  87  40 136   0 575   0   8   0]
 [  0   0   0   0   0   7   0 952   0  41]
 [  4   1   8   3   2   1   1   4 976   0]
 [  1   0   0   0   0   3   0  26   0 970]]
0.8748034992470057
```

Create the CNN Model

```
CNN_model = Sequential([
CNN_model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.2),
    Dense(32, activation='relu'),
    Dropout(0.2),
    Dense(10, activation='softmax')
])
CNN_model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics
```

```
CNN_model.fit(x_train, y_train, epochs=5)
```

```
Epoch 1/5
1875/1875 [==============================] - 39s 21ms/step - loss: 0.5733 - accu
Epoch 2/5
1875/1875 [==============================] - 39s 21ms/step - loss: 0.3484 - accu
Epoch 3/5
1875/1875 [==============================] - 39s 21ms/step - loss: 0.2979 - accu
Epoch 4/5
1875/1875 [==============================] - 39s 21ms/step - loss: 0.2695 - accu
Epoch 5/5
1875/1875 [==============================] - 39s 21ms/step - loss: 0.2474 - accu
<keras.callbacks.History at 0x7f86368f4a90>
```

```
y_pred_CNN = CNN_model.predict(x_test)
```

```
y_pred_CNN = np.argmax(y_pred_CNN, axis=1)
print(confusion_matrix(y_test, y_pred_CNN))
f1_score(y_test, y_pred_CNN, average='macro')
```

```
[[904   1  11  23   5   1  51   0   4   0]
 [  1 974   0  20   2   0   2   0   1   0]
 [ 23   1 860   9  66   0  40   1   0   0]
 [ 14   3   9 938  17   0  16   0   3   0]
 [  1   1  64  27 874   0  32   0   1   0]
 [  0   0   0   0   0 977   0  14   0   9]
 [167   0  69  36  80   0 634   0  14   0]
 [  0   0   0   0   0   6   0 973   0  21]
 [  2   3   0   5   1   5   4   4 975   1]
 [  0   0   0   0   0   5   1  27   0 967]]
0.9059275834044541
```

Differences between the models

In terms of accuracy, they were about the same although the CNN did perform marginally better than the RNN did. The CNN achieved an f1 score of 0.91 on the test data while the RNN achived an accuracy of 0.87. This is pretty impressive considering that the CNN was also faster to run. I ran 5 epochs on both models. For the CNN, each epoch took roughly 40 seconds, and totally took about 200 seconds. The RNN, on the other hand, took about 125 seconds per epoch, and the 5 epochs totally took about 625 seconds. In conclusion, the CNN performed marginally better in terms of accuracy but was able to train a lot faster.

RNN + CNN Combination

CNN are optimized for extracting unlabled features such as in an image. However, sometimes we want more than just the main object in an image. Multilabel classification is an important area in the field of image classification. For example, given a picture of a ship, we might want to classify the ship but also the water and the sky behind it. Traditionally, CNNs were used to complete this task. We could break up the image into multiple single label classification problems to find all of the classes inside of an image. The model fails when trying to model the dependency between classes however. This is where RNN can come in. RNN are good at finding relationships between sequences of data. For example, it is very common to see water in a picture of a ship, so when combining a CNN and an RNN, if a ship is found, it will also be more likely to classify the water. The way it works is that the CNN creates an image embedding vector and each label will have a label embedding vector. The RNN will then compute the probability of a label based on the miage embedding from the CNN and the output from the last iteration. It has many advantages such as: It is more compact and powerful, it can predict smaller objects with more need for context, and it can utilize semantic and co-occurence dependencies. As for the RNN, an LTSM model is used to combat the gradient vanishing/exploding issues that classic RNNs commonly face.