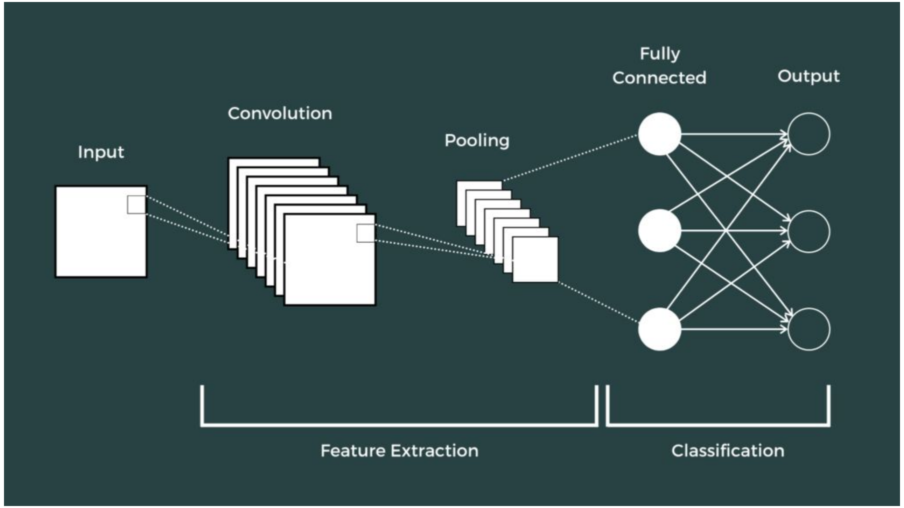
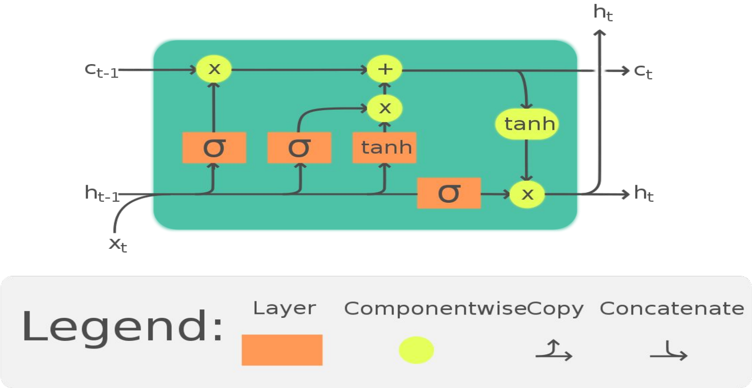
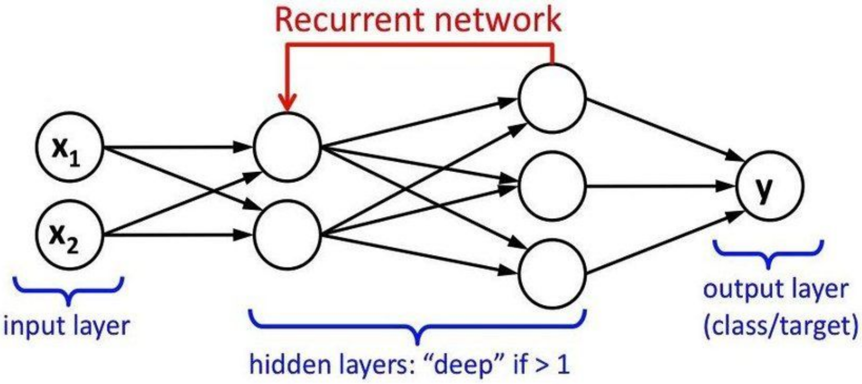
Handwritten Digits Recognition

Presented by Group 10
Mark Garcia, Alex Hwang, Hanson Nguyen
Andrew Phan, Anthony Reyes, and Linda Trinh

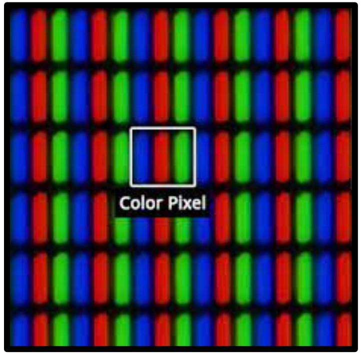


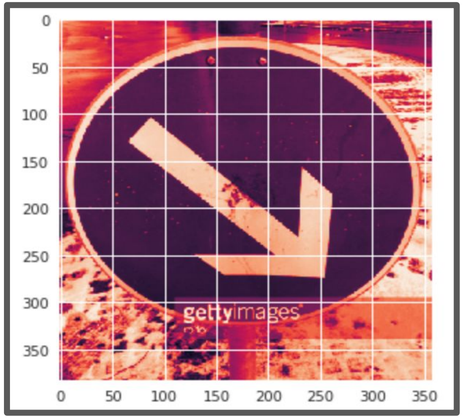


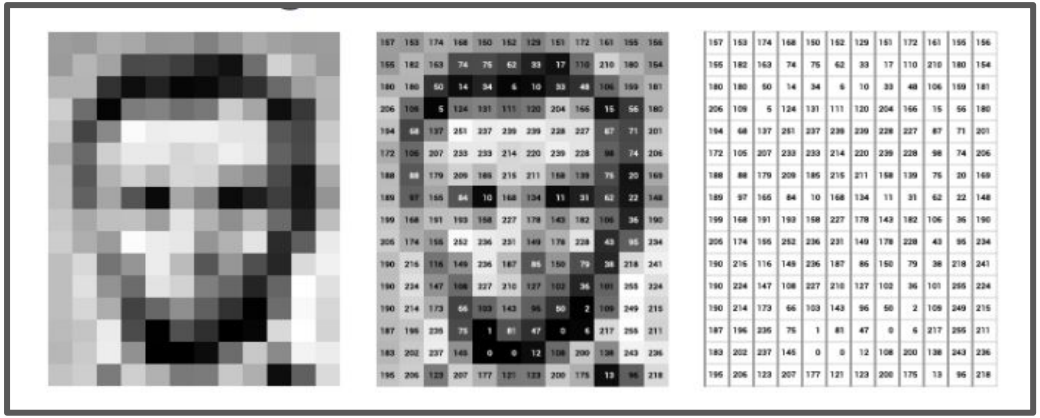


Data interpretation

- Machine doesn’t “see” images
- Images are a large array of numbers







157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
206	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	95	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

Data interpretation

- Conversion required for improve time and space
- Formula from Matplotlib library for conversion

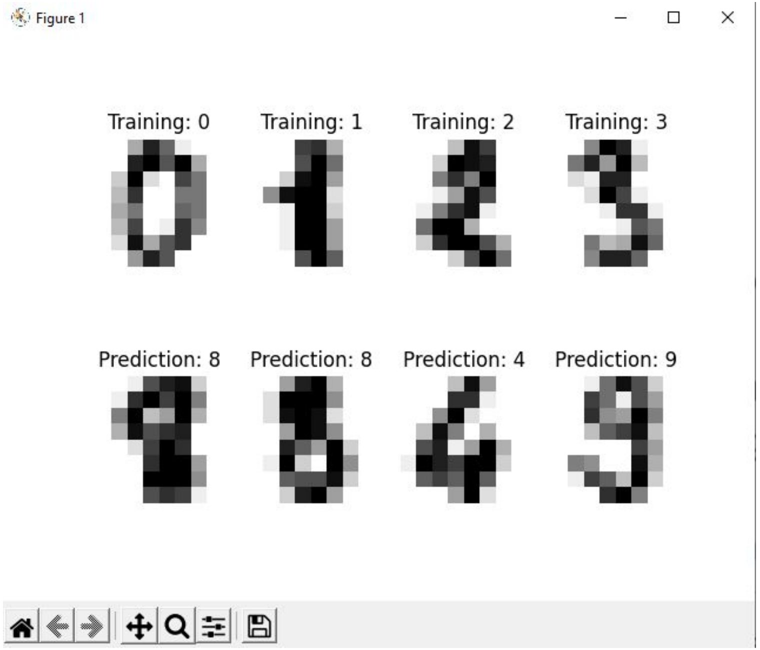
Gray Pixels = 0.2989 * R + 0.5870 * G + 0.1140 * B

- Picture converted into tensor of integers

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	29	150	195	254	255	254	176	193	150	96	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	48	166	224	253	253	234	196	253	253	253	253	233	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	93	244	249	253	187	46	10	8	4	10	194	253	253	233	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	107	253	253	230	48	0	0	0	0	0	192	253	253	156	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	3	20	20	15	0	0	0	0	0	43	224	253	245	74	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	249	253	245	126	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	14	101	223	253	248	124	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	11	166	239	253	253	253	187	30	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	16	248	250	253	253	253	253	232	213	111	2	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	43	98	98	208	253	253	253	253	187	22	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	51	119	253	253	253	76	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	183	253	253	139	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	182	253	253	104	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	85	249	253	253	36	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	214	253	253	173	11	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	98	247	253	253	226	9	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	42	150	252	253	253	233	53	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	42	115	42	60	115	159	240	253	253	250	175	25	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	187	253	253	253	253	253	253	253	197	86	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	103	253	253	253	253	253	232	67	1	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Data-to-training

- Support Vector Machines classifier, or the SVM for short.
- Takes in test data and training data.
- Scans the training data numbers first, then deciphers the test data numbers.

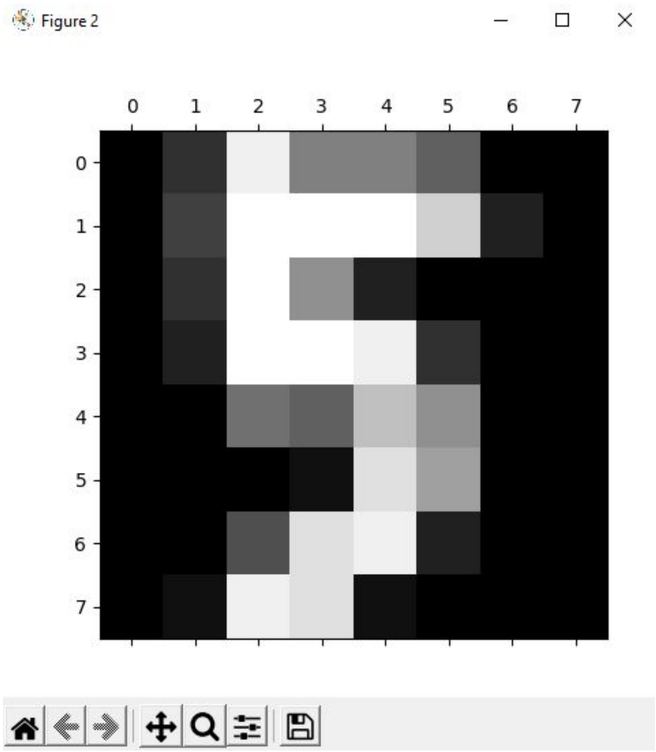


Data-to-training

- Program creates a pixelated graph for each number.
- Pixelated graphs become training material.

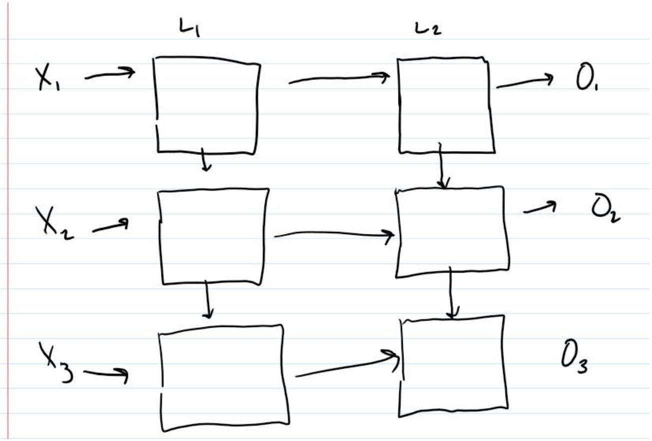
Classification report for classifier SVC(gamma=0.001):

	precision	recall	f1-score	support
0	1.00	0.99	0.99	88
1	0.99	0.97	0.98	91
2	0.99	0.99	0.99	86
3	0.98	0.87	0.92	91
4	0.99	0.96	0.97	92
5	0.95	0.97	0.96	91
6	0.99	0.99	0.99	91
7	0.96	0.99	0.97	89
8	0.94	1.00	0.97	88
9	0.93	0.98	0.95	92
accuracy			0.97	899
macro avg	0.97	0.97	0.97	899
weighted avg	0.97	0.97	0.97	899



Long Short-Term Memory (LSTM)

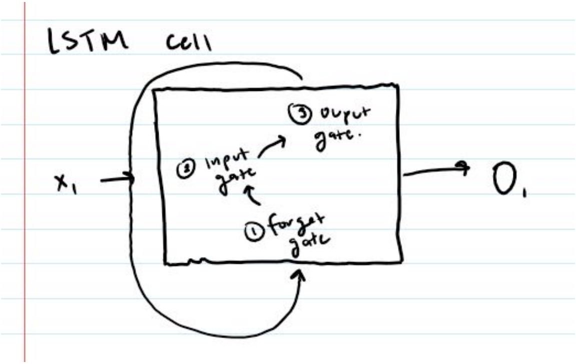
- Subcategory of recurrent neural network (RNN)
- Cells are most commonly composed of 3 gates (input, output, forget)
- Input data passes into one cell, is modified in the cell, and then is outputted to the next layer and also the next cell in the same layer.



- X: input
- L: layer
- O: output

LSTM (cont)

- Each cell of an LSTM layer does 4 things.
    - Decide what needs to be forgotten
    - Take in new input
    - Modify current cell state
    - Output the modified cell state

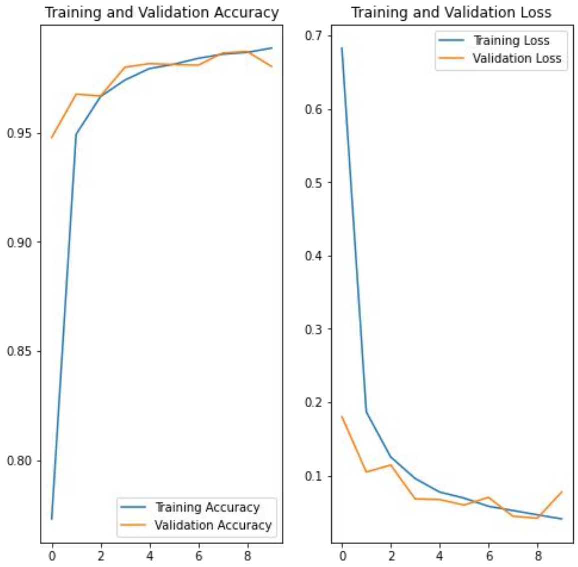


- The forget-gate decides what needs to be forgotten
- The input-gate takes in new information
- The output-gate filters the modified cell state and decides what information should be passed to the next nodes / layers

Our LSTM Model

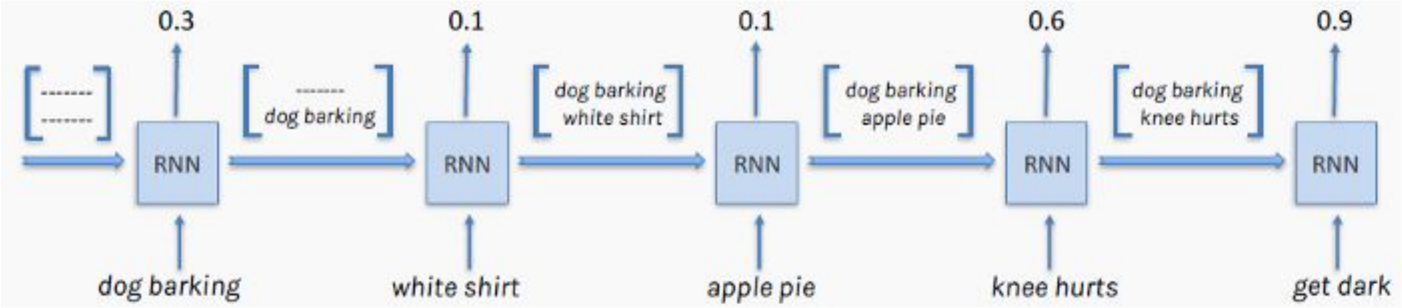
- Comprised of 2 LSTM layers, 2 Dense Layers, and Dropout layers in between each to help with over fitting.
- Results in an accuracy rating of 98.88% with a loss of .0411

```python
# RNN USING LSTM LAYERS
import numpy as np
import tensorflow as tf
from tensorflow import keras
from keras.layers import Dense, LSTM, Dropout
import cv2 as cv
import matplotlib.pyplot as plt

# import the mnist dataset that contains images of handwritten digits
(xTrain, yTrain), (xTest, yTest) = keras.datasets.mnist.load_data()

# normalize training and testing data to be btwn 0 - 1
xTrain = tf.keras.utils.normalize(xTrain, axis=1)
xTest = tf.keras.utils.normalize(xTest, axis=1)


model = tf.keras.models.Sequential()
# add LSTM layers to model followed by a dropout layer as my "output" layer
model.add(LSTM(128, input_shape=(xTrain.shape[1:]), activation='relu', return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(128, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(10, activation='softmax'))
# compile model
model.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True), optimizer='adam', metrics=['accuracy'])
# train out model with the training sets
history = model.fit(xTrain, yTrain, epochs=10, validation_data=(xTest, yTest))
```



Recurrent Neural Network (RNN)

- Artificial neural network
- Why is it important?
  - Many different use cases
  - Used in popular apps



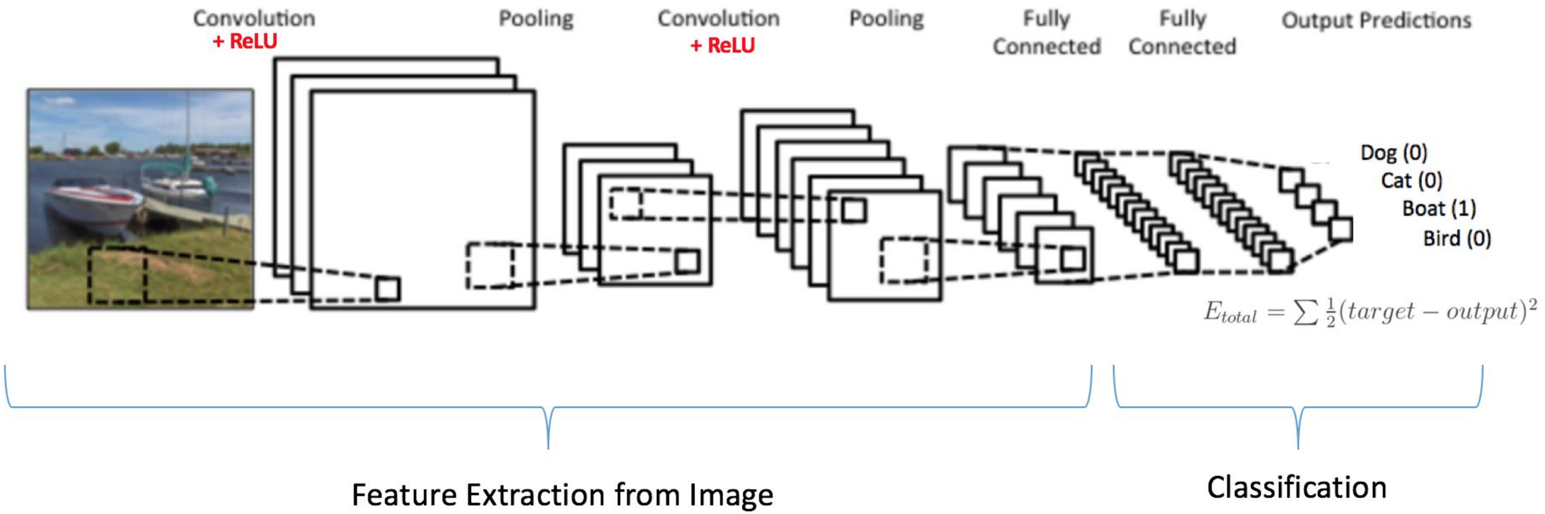
Our RNN Model

- Tensorflow
- 3 Dense layers, 1 Flatten layer
- 96.69% accuracy
- 0.1267 loss


```
database = tf.keras.datasets.mnist
(xTrain, yTrain),(xTest, yTest) = database.load_data()
xTrain = tf.keras.utils.normalize(xTrain, axis=1)
xTest = tf.keras.utils.normalize(xTest, axis=1)
#build our rnn model
model = tf.keras.models.Sequential()
# # add our layers
model.add(tf.keras.layers.Flatten(input_shape=(28, 28)))
model.add(tf.keras.layers.Dense(units=128, activation=tf.nn.relu))
model.add(tf.keras.layers.Dense(units=128, activation=tf.nn.relu))
#output layer
model.add(tf.keras.layers.Dense(units=10, activation=tf.nn.softmax))
#compile model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
epochs = 10
#train the rnn model and keep history of accuracy and loss
history = model.fit(xTrain, yTrain, epochs=epochs, validation_split=.3)
loss, accuracy = model.evaluate(xTest, yTest)
print("accuracy: ", accuracy)
print("loss: ", loss)
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs_range = range(epochs)
#plot the training and validation loss/accuracy~
plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')
plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```

Convolutional Neural Network (CNN)

- What is it commonly used for?
  - Image recognition
  - Classification
- Consist of three main layers:
  - Convolutional
  - Pooling
  - Fully-connected

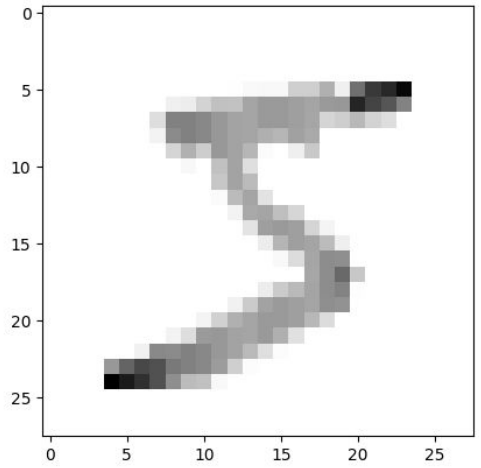


Our CNN Model

- Similar to RNN
  - TensorFlow
  - Split up datasets

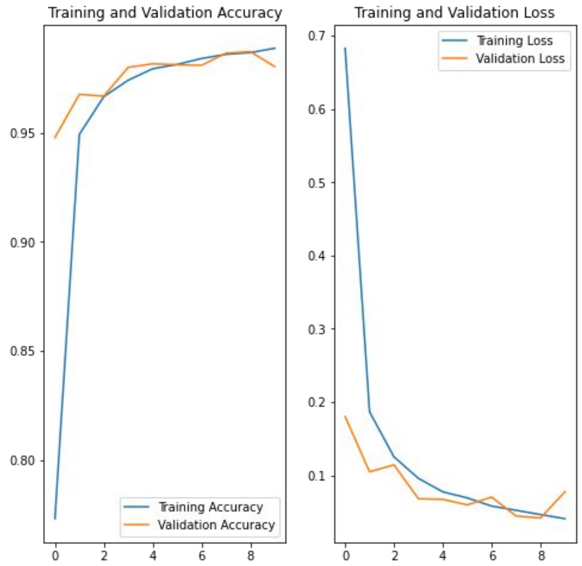
```python
# Train with convolutional neural networks (CNN)

# Import libraries
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt

# Load MNIST dataset
mnist = tf.keras.datasets.mnist
# Divide training and testing datasets: training = 60,000 & testing = 10,000
(xTrain, yTrain), (xTest, yTest) = mnist.load_data()


# Pre-process & Normalize data before CNN
xTrain = tf.keras.utils.normalize(xTrain, axis = 1)
xTest = tf.keras.utils.normalize(xTest, axis = 1)
plt.imshow(xTrain[0], cmap=plt.cm.binary)

x_trainr = np.array(xTrain).reshape(-1, 28, 28, 1)
x_testr = np.array(xTest).reshape(-1, 28, 28, 1)

# Build our CNN model
model = tf.keras.models.Sequential()

# Convolution Layer 1
model.add(tf.keras.layers.Conv2D(64, (3, 3), input_shape=x_trainr.shape[1:]))
model.add(tf.keras.layers.Activation("relu"))
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))

# Convolution Layer 2
model.add(tf.keras.layers.Conv2D(64, (3, 3)))
model.add(tf.keras.layers.Activation("relu"))
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))

# Convolution Layer 3
model.add(tf.keras.layers.Conv2D(64, (3, 3)))
model.add(tf.keras.layers.Activation("relu"))
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))

# Fully Connected Layer 1
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(64))
model.add(tf.keras.layers.Activation("relu"))
```

```python
# Fully Connected Layer 2
model.add(tf.keras.layers.Dense(32))
model.add(tf.keras.layers.Activation("relu"))

# Fully Connected Layer 3
model.add(tf.keras.layers.Dense(10))
model.add(tf.keras.layers.Activation("softmax"))

# Compile the model
model.compile(loss="sparse_categorical_crossentropy", optimizer='adam', metrics=['accuracy'])
epochs = 10
# Train the model
history = model.fit(x_trainr, yTrain, epochs=epochs, validation_split=0.3)

# Evaluate the model
loss, accuracy = model.evaluate(x_testr, yTest)
print("loss: ", loss)
print("accuracy: ", accuracy)

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```




```
loss:  0.06782113015651703
accuracy:  0.9829999804496765
```

Comparing Results



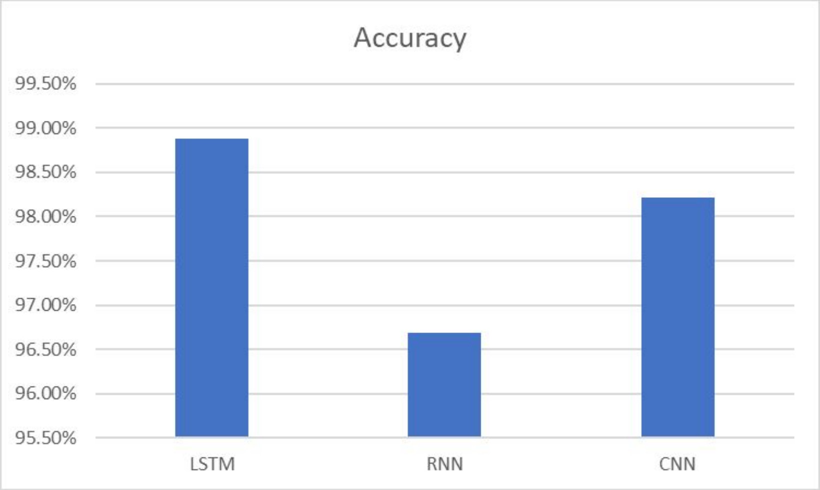
LSTM

RNN

CNN

Comparing Results

- LSTM
  - Accuracy: 98.88%
  - Loss: .0411
- CNN
  - Accuracy: 98.22%
  - Loss: .0641
- RNN
  - Accuracy: 96.69%
  - Loss: .1267



Thank you for listening!