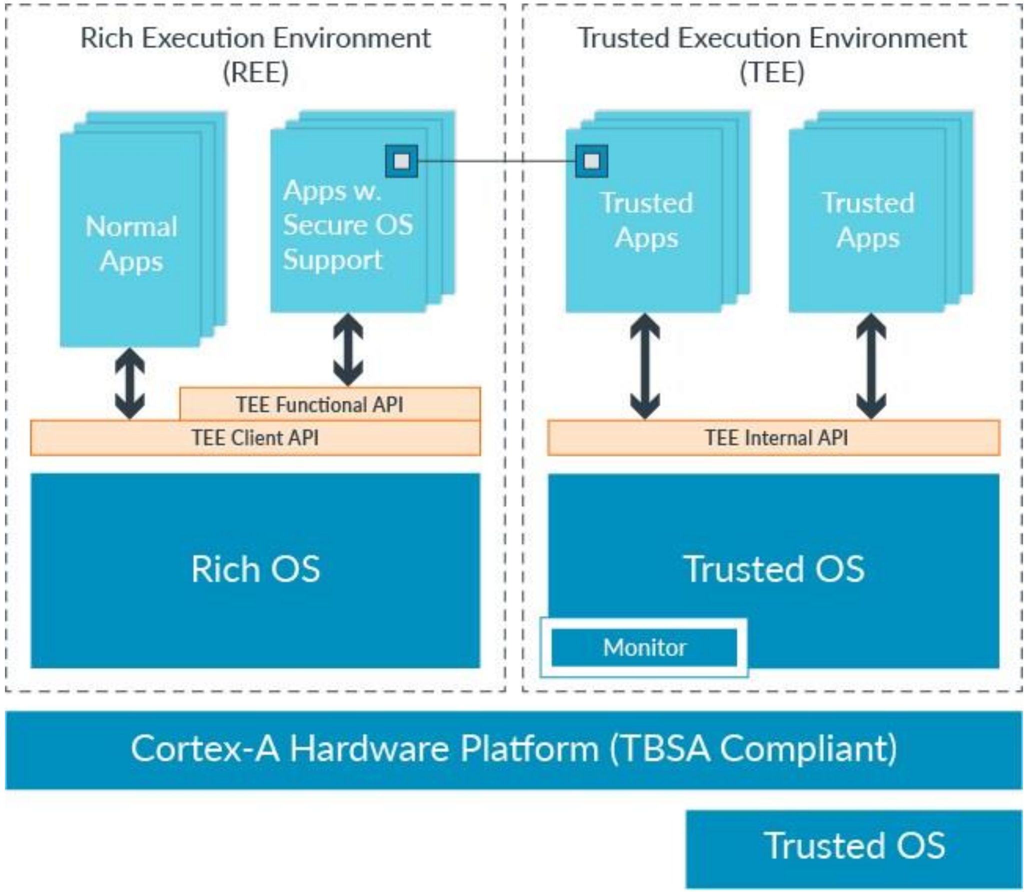
WEEK 6 UPDATE - ÉMPISTOS

Group 14 : Max Karen, AlRaheeq AlMaktum Al Rawas, Bradford Williams, Zack Wagner, and Anay Gulati

Sponsor: Md Tanvir Arafin

ARM TrustZone

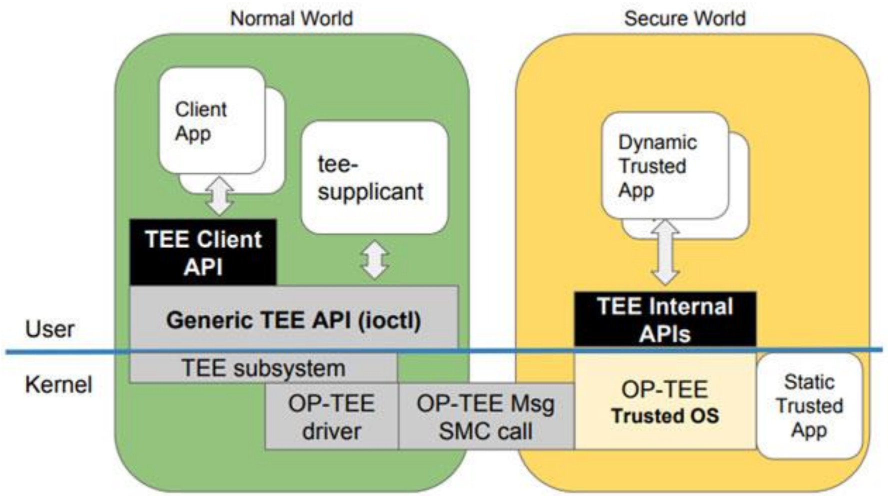
- Hardware-based security solution designed to create secure and non-secure execution environments on ARM-based processors.
- Seperates processor into Normal and Secure worlds
- Memory address space of Normal and Secure worlds through NS-bit which indicates the type of memory access
- Normal World can call Secure World through a Secure Monitor Call instruction



OP-TEE

"Trusted Execution Environment"

- Allows a trusted application to be ran in secure kernel world away from the non-secure OS
- Allows generic OS-level functions like Interrupt and thread handling, crypto services and shared memory
- How it works: A non-secure application calls the TEE API library, which then calls the host OS OP-TEE driver to send a request to the TEE to call a TA binary in the secure world to execute and return the result.



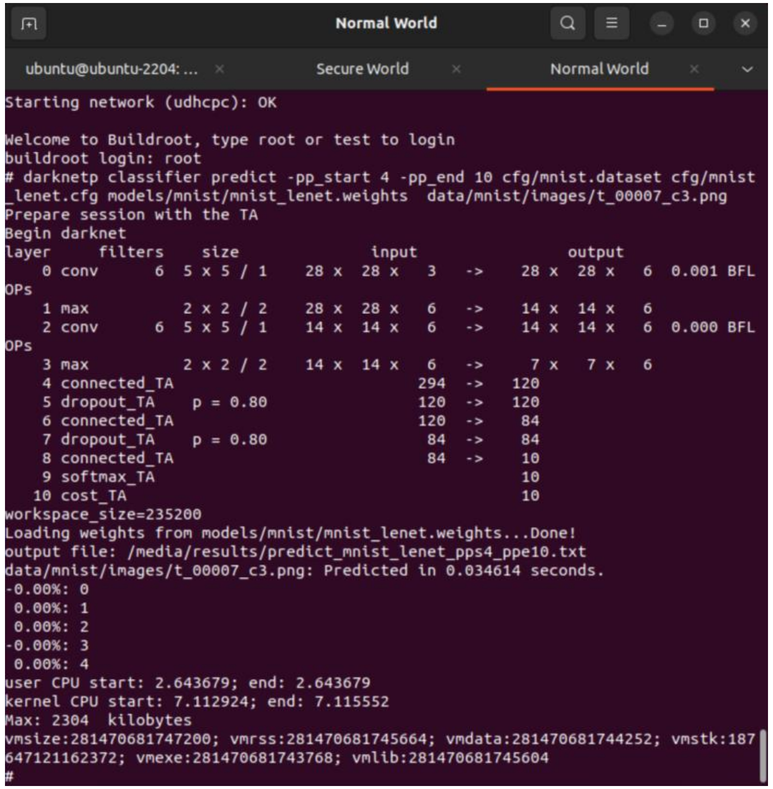
Darknet

"Open Source neural network framework written in C and CUDA"

- Optimized for speed, much faster than commonly used frameworks like TensorFlow due to reduced overhead of C programming.
- VERY Poor documentation
- Originally designed to run the YOLO object detection model

DARKNETZ

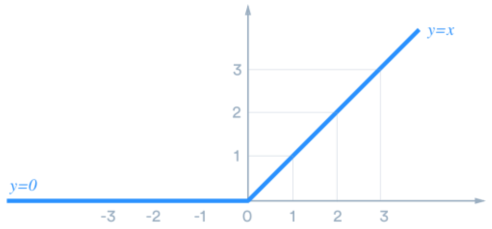
- DarkneTZ is an application that allows people to run multiple layers of a Deep Neural Network in ARM TrustZone. //Simplified version of Darknet, but is configured to take advantage of TrustZone.
- Allows for secure execution of neural network layers, particularly the final output layer, to execute in ARM TrustZone safely away from unsecure OS
    - Protects model and input data from outside adversarial attacks, such as power analysis to determine what the model is doing and poisoning the data to mess up accuracy of the model



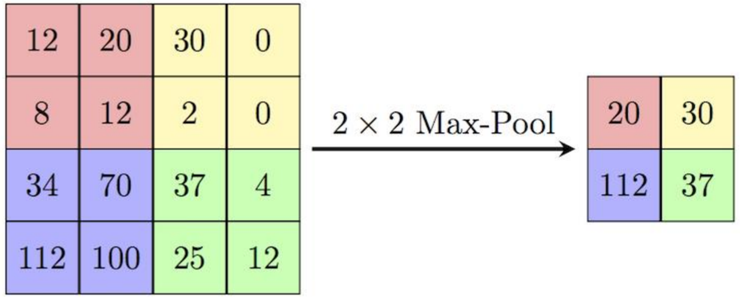
Deep Learning

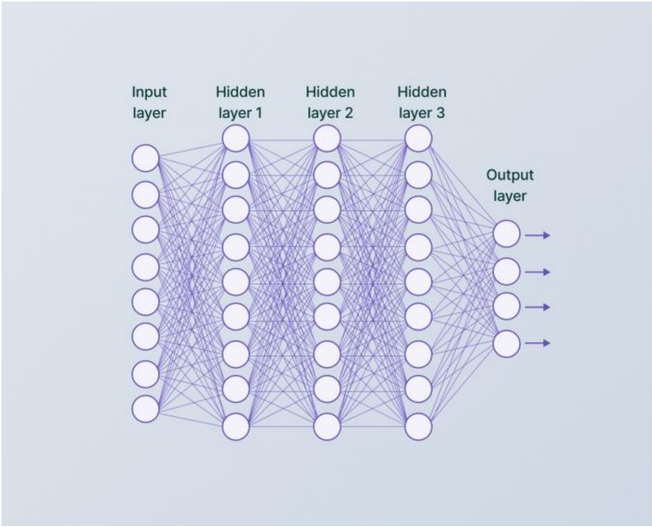
- Machine based learning on artificial neural networks to recognize complex patterns
- Made from layers of ‘neurons’ which learn to recognize patterns and predict/classify things in an image
- DarkneTZ helps prevent attacks on models by having some layers of a network execute in a trusted zone

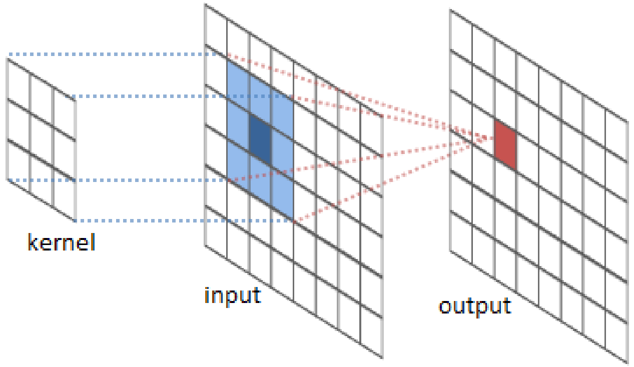
DEEP LEARNING : LAYERS

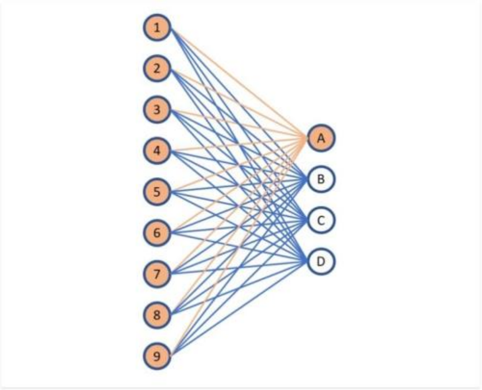


- Layers – different parts of a neural network that have different purposes
- Input Layer – input data
- Hidden Layers
  - Fully Connected layers
  - Convolutional layers
  - MaxPooling layers
- Output layer – final prediction





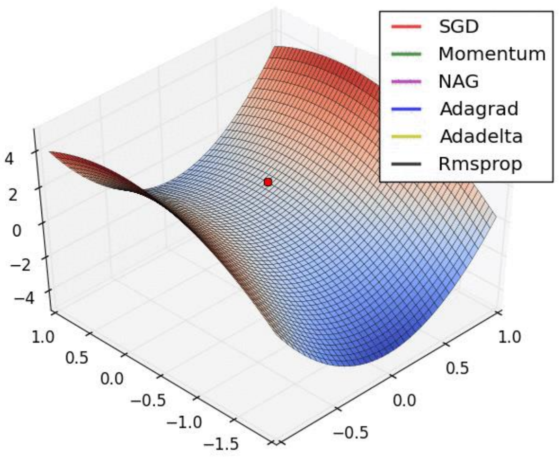




Deep Learning: the Learning Part

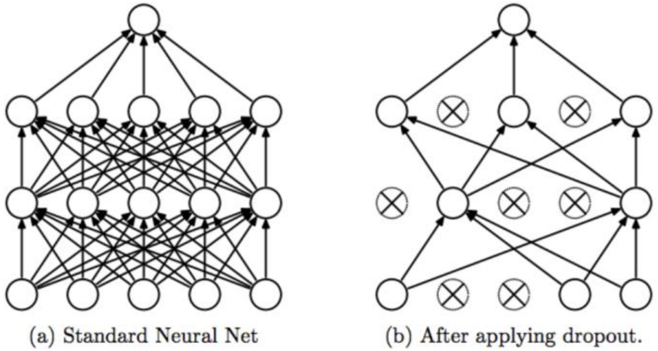
How a network learns is dependent on a few things:

- Learning rate
- Loss function
- Optimizer



Other important terms:

- Training vs Testing
- Underfitting vs Overfitting
- Dropout
- Data augmentation



KERAS & TENSORFLOW

- Keras is an API that is bundled with the TensorFlow library that allows for easy construction, modification, and testing of neural networks
- Designed to be human-readable
- Provides very easy modules and functions to implement a neural network.

Keras Example – Regular Neural Network

Importing tensorflow, keras, and our datasets

```
import tensorflow as tf
from tensorflow import keras

mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data() #x is data, y is labels
```

```
x_train = tf.keras.utils.normalize(x_train, axis = 1)
x_test= tf.keras.utils.normalize(x_test, axis = 1)
```

```
keras.utils.set_random_seed(20) #used to control seed of the model which is used to generate the weights and biases of

model = keras.models.Sequential([
    keras.layers.Input(shape = [28,28]),
    keras.layers.Flatten(),
    keras.layers.Dense(128, activation = tf.nn.relu), #adding hidden layers for feature recognition
    keras.layers.Dense(128, activation = tf.nn.relu),
    keras.layers.Dense(10, activation = tf.nn.softmax), #final output recognition layer. num of neurons is number of l
])
# another way to add layers outside of the constructor using add() function
# model.add(keras.layers.Flatten())
# model.add(keras.layers.Dense(128, activation = tf.nn.relu)) #adding hidden layers for feature recognition
# model.add(keras.layers.Dense(128, activation = tf.nn.relu))
# model.add(keras.layers.Dense(10, activation = tf.nn.softmax)) #final output recognition layer. num of neurons is num

#loss = keras.losses.SparseCategoricalCrossentropy(from_logits=true)
#optim = keras.optimizers.Adam(lr = 0.001)
#metrics = ['accuracy']
#model.compile(loss = loss, optimizer = optim, metrics = metrics)

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



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

```
Epoch 1/5
1875/1875 [==============================] - 3s 1ms/step - loss: 0.2580 - accuracy: 0.9258
Epoch 2/5
1875/1875 [==============================] - 2s 1ms/step - loss: 0.1038 - accuracy: 0.9683
Epoch 3/5
1875/1875 [==============================] - 3s 1ms/step - loss: 0.0720 - accuracy: 0.9773
Epoch 4/5
1875/1875 [==============================] - 3s 1ms/step - loss: 0.0519 - accuracy: 0.9834
Epoch 5/5
1875/1875 [==============================] - 2s 1ms/step - loss: 0.0400 - accuracy: 0.9872

<keras.src.callbacks.History at 0x2140e759c90>
```

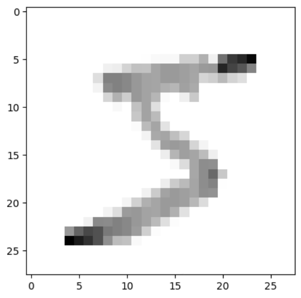
Creating neural network

Adding the optimizer and loss function

Training the model

```
import matplotlib.pyplot as plt

plt.imshow(x_train[0], cmap = plt.cm.binary)
plt.show()
print(x_train[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.        ]
 [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.         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.         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.         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.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.00393124 0.02332955 0.02620568 0.02625207 0.17420356 0.17566281
  0.28629534 0.05664824 0.51877786 0.71632322 0.77892406 0.89301644
  0.         0.         0.         0.        ]
```

Testing the model

```
print(model.summary())
print('\n')
val_loss, val_acc = model.evaluate(x_test, y_test, verbose = 2)
print(val_loss, val_acc)
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0

 dense (Dense)               (None, 128)               100480

 dense_1 (Dense)             (None, 128)               16512

 dense_2 (Dense)             (None, 10)                1290

=================================================================
Total params: 118282 (462.04 KB)
Trainable params: 118282 (462.04 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None


313/313 - 0s - loss: 0.0861 - accuracy: 0.9756 - 482ms/epoch - 2ms/step
0.08612360805273056 0.975600004196167
```

```
model.save('num_reader.model')
```

```
INFO:tensorflow:Assets written to: num_reader.model\assets
INFO:tensorflow:Assets written to: num_reader.model\assets
```

Resulting loss and accuracy

KERAS EXAMPLE – CONVOLUTIONAL NEURAL NETWORK

Creating convolutional neural network

```python
import tensorflow as tf
from tensorflow import keras
from keras import layers
import matplotlib.pyplot as plt
```

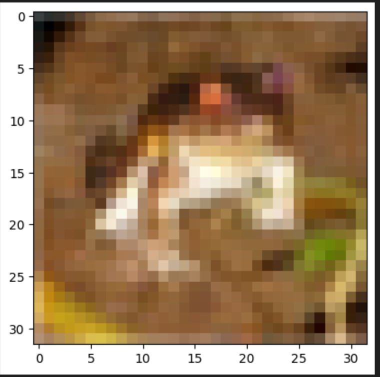
```python
#convolutional neural network (CNN)
cifar10 = keras.datasets.cifar10

(train_images, train_labels), (test_images, test_labels) = cifar10.load_data()
model2 = keras.models.Sequential([
    layers.Conv2D(32, kernel_size = (3,3), strides = (1,1), padding = 'same', activation = 'relu', input_shape = (32,
    layers.MaxPool2D(2, 2),
    layers.Conv2D(32, 3, activation = 'relu'),
    layers.MaxPool2D(2,2),
    layers.Flatten(),
    layers.Dense(64, activation = 'relu'),
    layers.Dense(10)
])

plt.imshow(train_images[0])
plt.show()

train_images = keras.utils.normalize(train_images, axis = 1)
test_images = keras.utils.normalize(test_images, axis = 1)

class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
```




```python
print(train_images.shape)
print(model2.summary())
```

```
(50000, 32, 32, 3)
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 32)	896
max_pooling2d (MaxPooling2D)	(None, 16, 16, 32)	0
conv2d_1 (Conv2D)	(None, 14, 14, 32)	9248
max_pooling2d_1 (MaxPooling2D)	(None, 7, 7, 32)	0
flatten_1 (Flatten)	(None, 1568)	0
dense_3 (Dense)	(None, 64)	100416
dense_4 (Dense)	(None, 10)	650

```
Total params: 111210 (434.41 KB)
Trainable params: 111210 (434.41 KB)
Non-trainable params: 0 (0.00 Byte)

None
```

Adding optimizer and loss function

```python
loss = keras.losses.SparseCategoricalCrossentropy(from_logits=True)
optim = keras.optimizers.Adam(learning_rate = 0.001)
metrics = ['accuracy']
model2.compile(loss = loss, optimizer = optim, metrics = metrics)
```

training

```python
model2.fit(train_images, train_labels, epochs = 20, batch_size = 32)
```

```
Epoch 1/20
1563/1563 [==============================] - 20s 13ms/step - loss: 1.5415 - accuracy: 0.4551
Epoch 2/20
1563/1563 [==============================] - 19s 12ms/step - loss: 1.2576 - accuracy: 0.5579
Epoch 3/20
1563/1563 [==============================] - 19s 12ms/step - loss: 1.1262 - accuracy: 0.6058
Epoch 4/20
1563/1563 [==============================] - 19s 12ms/step - loss: 1.0361 - accuracy: 0.6376
Epoch 5/20
1563/1563 [==============================] - 21s 13ms/step - loss: 0.9735 - accuracy: 0.6614
Epoch 6/20
1563/1563 [==============================] - 21s 13ms/step - loss: 0.9262 - accuracy: 0.6759
Epoch 7/20
1563/1563 [==============================] - 21s 14ms/step - loss: 0.8809 - accuracy: 0.6940
Epoch 8/20
1563/1563 [==============================] - 22s 14ms/step - loss: 0.8420 - accuracy: 0.7073
Epoch 9/20
1563/1563 [==============================] - 22s 14ms/step - loss: 0.8108 - accuracy: 0.7173
Epoch 10/20
1563/1563 [==============================] - 23s 14ms/step - loss: 0.7789 - accuracy: 0.7309
Epoch 11/20
1563/1563 [==============================] - 22s 14ms/step - loss: 0.7512 - accuracy: 0.7370
Epoch 12/20
1563/1563 [==============================] - 21s 13ms/step - loss: 0.7222 - accuracy: 0.7472
Epoch 13/20
...
Epoch 19/20
1563/1563 [==============================] - 21s 13ms/step - loss: 0.5731 - accuracy: 0.7965
Epoch 20/20
1563/1563 [==============================] - 21s 13ms/step - loss: 0.5569 - accuracy: 0.8032
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

<keras.src.callbacks.History at 0x2149cee6450>
```

```python
model2.evaluate(test_images, test_labels, batch_size = 32)
#print(train_images[0])
#plt.imshow(test_images[0])
#plt.show()
```

```
313/313 [==============================] - 2s 5ms/step - loss: 1.1264 - accuracy: 0.6629

[1.1263961791992188, 0.6628999710083008]
```

Testing the model

Goal Going Forward

- Learn from Keras and implement an easy-to-use Python frontend to construct and train neural networks in C that can then be securely executed using DarkneTZ
- Demonstrate basic models functioning within DarkneTZ on the Raspberry Pi 3B+
- Next task: examine darknet code and learn how to create a basic network with c instead of tensorflow