In [ ]:
```
from keras.datasets import mnist

data = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.
npz
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
```

In [ ]:
```
data

type(data)
```

Out[ ]:
```
tuple
```

In [ ]:
```
(X_train, y_train), (X_test, y_test) = data
```

In [ ]:
```
X_train[0].shape
```

Out[ ]:
```
(28, 28)
```

In [ ]:
```
X_train.shape
```

Out[ ]:
```
(60000, 28, 28)
```

In [ ]:
```
X_train = X_train.reshape((X_train.shape[0], 28*28)).astype('float32')
X_test = X_test.reshape((X_test.shape[0], 28*28)).astype('float32')
```

In [ ]:
```
X_train = X_train / 255
X_test = X_test / 255
```

In [ ]:
```
from keras.utils import np_utils

print(y_test.shape)

y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)

num_classes = y_test.shape[1]

print(y_test.shape)
```

```
(10000,)
(10000, 10)
```

In [ ]:
```
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()
```

```
model.add(Dense(32, input_dim = 28 * 28, activation= 'relu'))

model.add(Dense(64, activation = 'relu'))

model.add(Dense(10, activation  = 'softmax'))
```

In [ ]:
```
model.compile(loss = 'categorical_crossentropy', optimizer = 'adam', metrics = ['accura
```

In [ ]:
```
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 32)                25120
_________________________________________________________________
dense_2 (Dense)              (None, 64)                2112
_________________________________________________________________
dense_3 (Dense)              (None, 10)                650
=================================================================
Total params: 27,882
Trainable params: 27,882
Non-trainable params: 0
_________________________________________________________________
```

ANN with categorical_crossentropy and adam

In [ ]:
```
model.compile(loss = 'categorical_crossentropy', optimizer = 'adam', metrics = ['accura
```

In [ ]:
```
model.fit(X_train, y_train, epochs= 10, batch_size = 100)
```

```
Epoch 1/10
600/600 [==============================] - 3s 3ms/step - loss: 0.4227 - accuracy: 0.8794
Epoch 2/10
600/600 [==============================] - 2s 3ms/step - loss: 0.1941 - accuracy: 0.9437
Epoch 3/10
600/600 [==============================] - 2s 3ms/step - loss: 0.1522 - accuracy: 0.9549
Epoch 4/10
600/600 [==============================] - 2s 3ms/step - loss: 0.1265 - accuracy: 0.9632
Epoch 5/10
600/600 [==============================] - 2s 3ms/step - loss: 0.1091 - accuracy: 0.9676
Epoch 6/10
600/600 [==============================] - 2s 3ms/step - loss: 0.0963 - accuracy: 0.9709
Epoch 7/10
600/600 [==============================] - 2s 3ms/step - loss: 0.0860 - accuracy: 0.9743
Epoch 8/10
600/600 [==============================] - 2s 3ms/step - loss: 0.0785 - accuracy: 0.9756
Epoch 9/10
600/600 [==============================] - 2s 3ms/step - loss: 0.0698 - accuracy: 0.9788
Epoch 10/10
600/600 [==============================] - 2s 3ms/step - loss: 0.0648 - accuracy: 0.9800
```

Out[ ]:
```
<keras.callbacks.History at 0x7f9cb3dc5bd0>
```

Ann with RMSprop and categorical_crossentropy

In [ ]:
```
model.compile(loss = 'categorical_crossentropy', optimizer = 'adam', metrics = ['accura
```

In [ ]:
```
model.fit(X_train, y_train, epochs= 10, batch_size = 100)
```

```
Epoch 1/10
600/600 [==============================] - 3s 4ms/step - loss: 0.0603 - accuracy: 0.9813
Epoch 2/10
600/600 [==============================] - 2s 3ms/step - loss: 0.0541 - accuracy: 0.9828
Epoch 3/10
600/600 [==============================] - 2s 4ms/step - loss: 0.0496 - accuracy: 0.9846
Epoch 4/10
600/600 [==============================] - 2s 3ms/step - loss: 0.0446 - accuracy: 0.9857
Epoch 5/10
600/600 [==============================] - 2s 3ms/step - loss: 0.0415 - accuracy: 0.9869
Epoch 6/10
600/600 [==============================] - 2s 3ms/step - loss: 0.0382 - accuracy: 0.9880
Epoch 7/10
600/600 [==============================] - 2s 3ms/step - loss: 0.0362 - accuracy: 0.9887
Epoch 8/10
600/600 [==============================] - 2s 3ms/step - loss: 0.0337 - accuracy: 0.9890
Epoch 9/10
600/600 [==============================] - 2s 3ms/step - loss: 0.0294 - accuracy: 0.9909
Epoch 10/10
600/600 [==============================] - 2s 3ms/step - loss: 0.0293 - accuracy: 0.9906
```

Out[ ]:
```
<keras.callbacks.History at 0x7f9cb24ef3d0>
```

ANN with Poisson and Adam

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

In [ ]:
```
model.fit(X_train, y_train, epochs= 10, batch_size = 100)
```

```
Epoch 1/10
600/600 [==============================] - 2s 3ms/step - loss: 0.1028 - accuracy: 0.9909
Epoch 2/10
600/600 [==============================] - 2s 3ms/step - loss: 0.1024 - accuracy: 0.9924
Epoch 3/10
600/600 [==============================] - 2s 3ms/step - loss: 0.1024 - accuracy: 0.9922
Epoch 4/10
600/600 [==============================] - 2s 4ms/step - loss: 0.1022 - accuracy: 0.9932
Epoch 5/10
600/600 [==============================] - 2s 4ms/step - loss: 0.1021 - accuracy: 0.9931
Epoch 6/10
600/600 [==============================] - 2s 3ms/step - loss: 0.1017 - accuracy: 0.9949
Epoch 7/10
600/600 [==============================] - 2s 3ms/step - loss: 0.1017 - accuracy: 0.9946
Epoch 8/10
600/600 [==============================] - 2s 3ms/step - loss: 0.1017 - accuracy: 0.9944
Epoch 9/10
600/600 [==============================] - 2s 3ms/step - loss: 0.1016 - accuracy: 0.9949
Epoch 10/10
600/600 [==============================] - 2s 3ms/step - loss: 0.1014 - accuracy: 0.9953
```

Out[ ]:
```
<keras.callbacks.History at 0x7f9cb39f5090>
```

ANN with Poisson and RMSprop

In [ ]:
```
model.compile(loss = 'Poisson', optimizer = 'RMSprop', metrics = ['accuracy'])
```

In [ ]:
```
model.fit(X_train, y_train, epochs= 10, batch_size = 100)
```

```
Epoch 1/10
600/600 [==============================] - 3s 3ms/step - loss: 0.1012 - accuracy: 0.9959
Epoch 2/10
600/600 [==============================] - 2s 3ms/step - loss: 0.1011 - accuracy: 0.9965
Epoch 3/10
600/600 [==============================] - 2s 3ms/step - loss: 0.1011 - accuracy: 0.9964
Epoch 4/10
600/600 [==============================] - 2s 3ms/step - loss: 0.1010 - accuracy: 0.9969
Epoch 5/10
600/600 [==============================] - 2s 3ms/step - loss: 0.1009 - accuracy: 0.9972
Epoch 6/10
600/600 [==============================] - 2s 3ms/step - loss: 0.1009 - accuracy: 0.9974
Epoch 7/10
600/600 [==============================] - 2s 3ms/step - loss: 0.1009 - accuracy: 0.9974
Epoch 8/10
600/600 [==============================] - 2s 3ms/step - loss: 0.1008 - accuracy: 0.9975
Epoch 9/10
600/600 [==============================] - 2s 3ms/step - loss: 0.1007 - accuracy: 0.9978
Epoch 10/10
600/600 [==============================] - 2s 3ms/step - loss: 0.1007 - accuracy: 0.9977
```

Out[ ]:
```
<keras.callbacks.History at 0x7f9cb257a8d0>
```

CNN

In [ ]:
```
# load mnist dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# compute the number of labels
num_labels = len(np.unique(y_train))

# convert to one-hot vector
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

# input image dimensions
image_size = x_train.shape[1]
# resize and normalize
x_train = np.reshape(x_train,[-1, image_size, image_size, 1])
x_test = np.reshape(x_test,[-1, image_size, image_size, 1])
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255
```

In [ ]:
```
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Activation, Dense, Dropout
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten
from tensorflow.keras.utils import to_categorical, plot_model

input_shape = (28, 28, 1)
batch_size = 128
kernel_size = 3
pool_size = 2
filters = 64
```

```
dropout = 0.2



model = Sequential()
model.add(Conv2D(filters=filters,
                 kernel_size=kernel_size,
                 activation='relu',
                 input_shape=input_shape))
model.add(MaxPooling2D(pool_size))
model.add(Conv2D(filters=filters,
                 kernel_size=kernel_size,
                 activation='relu'))
model.add(MaxPooling2D(pool_size))
model.add(Conv2D(filters=filters,
                 kernel_size=kernel_size,
                 activation='relu'))
model.add(Flatten())
# dropout added as regularizer
model.add(Dropout(dropout))
# output layer is 10-dim one-hot vector
model.add(Dense(10))
model.add(Activation('softmax'))
model.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_13 (Conv2D)           (None, 26, 26, 64)        640
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 13, 13, 64)        0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 11, 11, 64)        36928
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 3, 3, 64)          36928
_________________________________________________________________
flatten_4 (Flatten)          (None, 576)               0
_________________________________________________________________
dropout_4 (Dropout)          (None, 576)               0
_________________________________________________________________
dense_6 (Dense)              (None, 10)                5770
_________________________________________________________________
activation_3 (Activation)    (None, 10)                0
=================================================================
Total params: 80,266
Trainable params: 80,266
Non-trainable params: 0
_________________________________________________________________
```

Out[ ]:
```
'\nmodel = Sequential(\n    [\n        keras.Input(shape=input_shape),\n        layers.C
onv2D(32, kernel_size=(3, 3), activation="relu"),\n        layers.MaxPooling2D(pool_size
=(2, 2)),\n        layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),\n        la
yers.MaxPooling2D(pool_size=(2, 2)),\n        layers.Flatten(),\n        layers.Dropout
(0.5),\n        layers.Dense(num_classes, activation="softmax"),\n    ]\n)\n'
```

CNN with categorical_crossentropy and adam

In [ ]:
```
model.compile(loss='categorical_crossentropy',
```

```
              optimizer='adam',
              metrics=['accuracy'])
```

In [ ]:
```
# train the network
model.fit(x_train, y_train, epochs=5, batch_size=batch_size)
```

```
Epoch 1/5
469/469 [==============================] - 70s 148ms/step - loss: 0.0238 - accuracy: 0.9
924
Epoch 2/5
469/469 [==============================] - 69s 148ms/step - loss: 0.0181 - accuracy: 0.9
943
Epoch 3/5
469/469 [==============================] - 69s 148ms/step - loss: 0.0161 - accuracy: 0.9
950
Epoch 4/5
469/469 [==============================] - 69s 146ms/step - loss: 0.0153 - accuracy: 0.9
948
Epoch 5/5
469/469 [==============================] - 68s 146ms/step - loss: 0.0116 - accuracy: 0.9
959
```

Out[ ]:
```
<keras.callbacks.History at 0x7efdcb890490>
```

CNN with RMSprop and categorical_crossentropy

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

In [ ]:
```
# train the network
model.fit(x_train, y_train, epochs=5, batch_size=batch_size)
```

```
Epoch 1/10
600/600 [==============================] - 2s 2ms/step - loss: 0.0598 - accuracy: 0.9811
Epoch 2/10
600/600 [==============================] - 1s 2ms/step - loss: 0.0559 - accuracy: 0.9824
Epoch 3/10
600/600 [==============================] - 1s 2ms/step - loss: 0.0522 - accuracy: 0.9843
Epoch 4/10
600/600 [==============================] - 1s 2ms/step - loss: 0.0497 - accuracy: 0.9839
Epoch 5/10
600/600 [==============================] - 1s 2ms/step - loss: 0.0471 - accuracy: 0.9851
Epoch 6/10
600/600 [==============================] - 1s 2ms/step - loss: 0.0448 - accuracy: 0.9858
Epoch 7/10
600/600 [==============================] - 1s 2ms/step - loss: 0.0424 - accuracy: 0.9868
Epoch 8/10
600/600 [==============================] - 1s 2ms/step - loss: 0.0405 - accuracy: 0.9870
Epoch 9/10
600/600 [==============================] - 1s 2ms/step - loss: 0.0381 - accuracy: 0.9879
Epoch 10/10
600/600 [==============================] - 1s 2ms/step - loss: 0.0363 - accuracy: 0.9886
```

Out[ ]:
```
<keras.callbacks.History at 0x7f16c1367550>
```