Neural Networks

Hoyt Thomas

3/12/20

I. Abstract

To classify the fashion minst data set, two different types of Neural Networks are developed and tested to see the accuracy. A fully connected (Dense) Neural Network and a Convulutional Neural Network will be used.

II. Introduction

Neural Networks are a machine learning tool used for classification, regression and dimensionality reduction. Neural Networks were developed decades ago but have recently risen in popularity due to increased computational power paired with vast new data set available. They are meant to mirror the natural design of neurons in our brains, the input neurons are connected in to output layers through a variable number of hidden layers between the two.

Fully-connected neural networks and convolutional neural networks will be used to classify the Fashion_Minst data set, a set of 28 by 28 images of different fashion pieces classified into ten different categories. The fashion_minst data will be used to train a neural network with the goal of achieving 90% accuracy on the test data.

III. Theoretical Background

Fully-connected neural networks are have each neuron in the layer connected every neuron in the next layer each connection has weight associated with it, there is also a constant bias with each layer. Each input neuron can be fit into a vector, x, the wights into a matrix, A, and the biases into a vector, b. This allows to fit them into a familiar Ax + b format which will be multiplied by an activation function, σ. The purpose of the activation function is provide a separation between outputs so that they can classified into desired outputs. The activation function that will be used in this neural network is the Rectified Liner Unit (ReLU) which is defined as:

$$\sigma(x) = \max(x, 0) \text{EQ. 1}$$

where x is Ax+b. Each layer has a width which is the number of the neurons in the layer and the number of layers in the network is the depth of the network. For this classification the output needs to be a probability. For this the softmax function will be used, which is defined as:

$$p = 1\frac{1}{\sum_{j=1}^{m} e^{yj}}(e^{y_m}) \text{ EQ 2}$$

where y is the solution to the activation function and m is number of outputs. This step is also called the loss function.

Convolutional Neural Networks (CNN) have similar architecture to fully connected layers but instead of neurons being connected to all in subsequent layer, neurons are layered on

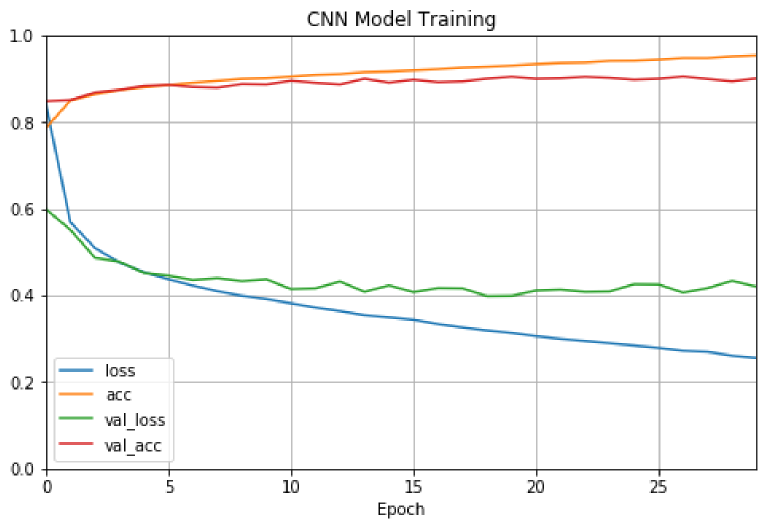
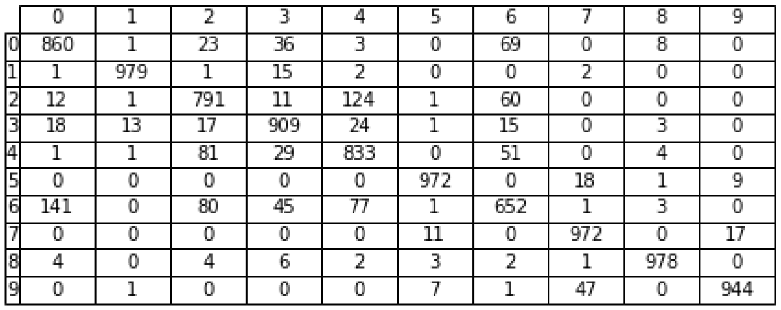


Figure 1: The accuracy and loss of the dense neural network against training and validation data

top of each other. The lowest level neurons only recognize the most simple trends (horizontal, vertical lines) and pass them as the input to the next layer which are capable of recognizing more complex trends. Layers recognize trends by passing over the image with a filter, to create a feature map which is the input for the next layer. Activation functions are used to create the feature maps, for this CNN will use the hyperbolic tangent function, which is defined as:

$$y = \tanh(x) \text{ EQ. 3}$$

The filters can be different sizes and move across images with different size steps or 'stride.' A large stride will decrease the size of the feature map since more of the image is in the filter during each step. To prevent missing information on the edges of the images, they are zero-padded so the filters can center on the edge pixel. Convolutional layers will often produce multiple feature maps which can quickly increase the computational load of the network. To curb this problem, feature maps are subsampled after each convolutional layer through average pooling, where all the values in the in a certain grid are averaged to produce a single value.



	0	1	2	3	4	5	6	7	8	9
0	860	1	23	36	3	0	69	0	8	0
1	1	979	1	15	2	0	0	2	0	0
2	12	1	791	11	124	1	60	0	0	0
3	18	13	17	909	24	1	15	0	3	0
4	1	1	81	29	833	0	51	0	4	0
5	0	0	0	0	0	972	0	18	1	9
6	141	0	80	45	77	1	652	1	3	0
7	0	0	0	0	0	11	0	972	0	17
8	4	0	4	6	2	3	2	1	978	0
9	0	1	0	0	0	7	1	47	0	944

Figure 2: Results of Neural Network against test data, x-axis is the target class and y-axis is the output class

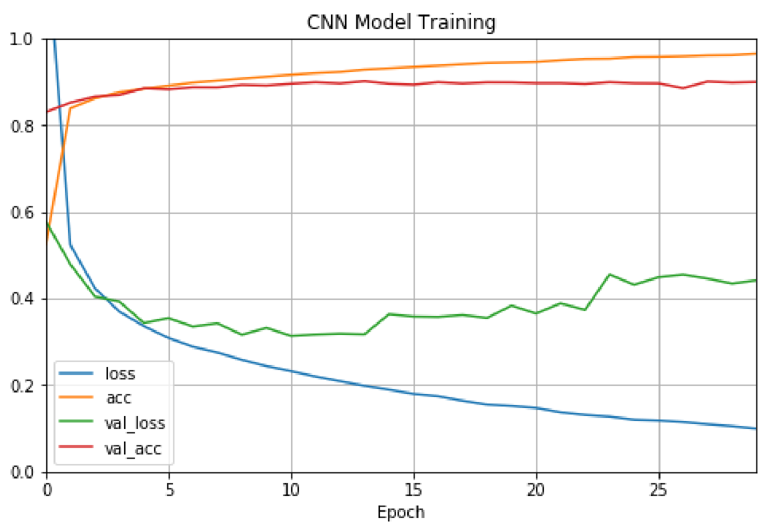


Figure 3: The accuracy and loss of the convolutional neural network against training and validation data

Neural Networks require three data sets to test: training, validation, and test data. The training data is used to train the model and determine necessary weights biases. The validation data is used to ensure the Network is not overfitting to the training data, or when the network develops a solution so fitted to the training data it will not classify the validation data well. This can happen when there are more parameters relative to the amount of data available. The test data is used to test the accuracy of the network

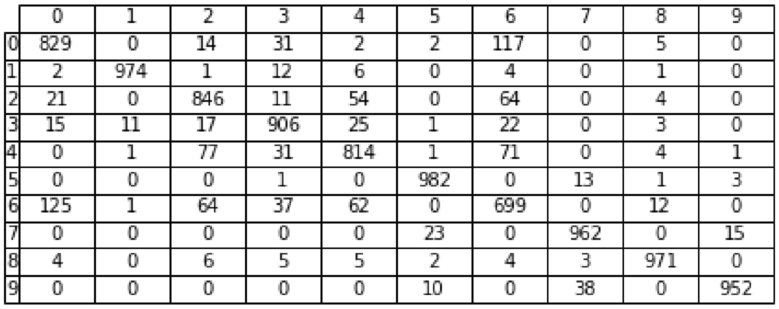
IV. Algorithm Development

Fully Connected Neural Network

To start, import the necessary modules and load in the fashion minst dataset. The minst data comes as uint8 data, pixel values range from 0 to 255 so to normalize all the data divide all the data by 255 so it is 0 to 1. The training data contains 60,000 images, partition 5,000 images to use as validation data so there as 55,000 training images and 5,000 validation images. Begin constructing the architecture of the network by defining a dense (fully connected) layer with a Relu (EQ. 1) activation function and a regularizer to curb some overfitting. Initialize the network as a sequential network using Tensorflow, then flatten the images [28X28] into a single vector. Build the neural network by creating multiple dense layers of same width, then add a few layers with decreasing width with the final width equaling the number of desired outputs. Repeat the narrowest layers a few times before feeding it into a layer with a softmax (EQ. 2) activation function. Compile the network using a sparse categorical cross-entropy loss function and define the learning rate (how much the model adjusts on each epoch). Train the model using test and validation data and graph the accuracy and loss (Fig. 1). Test the model against the test data and produce a confusion matrix (Fig. 2) to see where the model went wrong.

Convolutional Neural Network

To start, import the necessary modules and load in the fashion minst dataset. The minst data comes as uint8 data, pixel values range from 0 to 255 so to normalize all the data divide all the



	0	1	2	3	4	5	6	7	8	9
0	829	0	14	31	2	2	117	0	5	0
1	2	974	1	12	6	0	4	0	1	0
2	21	0	846	11	54	0	64	0	4	0
3	15	11	17	906	25	1	22	0	3	0
4	0	1	77	31	814	1	71	0	4	1
5	0	0	0	1	0	982	0	13	1	3
6	125	1	64	37	62	0	699	0	12	0
7	0	0	0	0	0	23	0	962	0	15
8	4	0	6	5	5	2	4	3	971	0
9	0	0	0	0	0	10	0	38	0	952

Figure 4: Results of the CNN against test data, x-axis is the target class and y-axis is the output class

data by 255 so it is 0 to 1. The training data contains 60,000 images, partition 5,000 images to use as validation data so there as 55,000 training images and 5,000 validation images. Begin constructing the architecture of the network by defining a dense (fully connected) layer with a hyperbolic tangent (EQ. 3) activation function and a regularizer to curb some overfitting. Define a convolutional layer with a hyperbolic tangent (EQ. 3) activation function and no zero-padding. Initialize the network as a sequential network using Tensorflow, and use the pre-defined convolutional layer with 6 filters and 5x5 filter size and stride of 5, since this is the first layer we want to zero-pad the image so feature maps are the same size of the original image. Then add an average pooling layer with kernel size of 2x2 and stride of 2. Add another convolutional layer with 36 filters and the same kernel size and stride length. Add another average pooling layer with the same kernel size and stride length. Add another convolutional layer with 108 filters and the same kernel size and stride length. Flatten the outputs to pass into a dense section of the network, add in four layers of the pre-defined dense layer with decreasing width. Add a final layer with a softmax activation function (EQ. 2). Compile the network using a sparse categorical cross-entropy loss function and define the learning rate (how much the model adjusts on each epoch). Train the model using test and validation data and graph the accuracy and loss (Fig. 3). Test the model against the test data and produce a confusion matrix (Fig. 4) to see where the model went wrong.

V. Computational Results

The fully-connected network achieved an accuracy of 88.9% and the convolutional network achieved an accuracy of 89.35%.

VI. Conclusion

The CNN performed better the dense NN and training the model used less computational power.

Appendix A: Python Commands

Tensorflow.keras.layers() -Dense, Conv2D, and AveragePooling2D

Creates neural network layer with specficed width, regularization for dense and kernel size, stride length for Conv2D and AveragePooling2D

Tensorflow.keras.models() – sequential

Initializes neural network

Tensorflow.keras.models.compile – loss, optimizers, and learning rate

Compiles model with given loss function, optimization, and learning rate

Appendix B: Python Code

See Attached PDF

In [4]:
```
import matplotlib.pyplot as plt
import numpy as np
import math
import pandas as pd
import tensorflow as tf
from sklearn.metrics import confusion_matrix
```

In [5]:
```
fashion_mnist = tf.keras.datasets.fashion_mnist
(X_train_full, y_train_full), (X_test, y_test) = fashion_mnist.load_data()
```

In [6]:
```
X_train_full.shape
```

Out[6]:
```
(60000, 28, 28)
```

In [7]:
```
plt.figure()
for k in range(9):
    plt.subplot(3,3,k+1)
    plt.imshow(X_train_full[k], cmap="gray")
    plt.axis('off')
plt.show()
```

In [8]:
```
X_valid = X_train_full[:5000] / 255.0
X_train = X_train_full[5000:] / 255.0
X_test = X_test / 255.0

y_valid = y_train_full[:5000]
y_train = y_train_full[5000:]
```

In [18]:
```
from functools import partial

my_dense_layer = partial(tf.keras.layers.Dense, activation="relu", kernel_regularizer=tf.keras.regularizers.l
2(0.000001))

model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=[28, 28]),
    my_dense_layer(750),
    my_dense_layer(750),
    my_dense_layer(750),
    my_dense_layer(500),
    my_dense_layer(250),
    my_dense_layer(125),
    my_dense_layer(75),
    my_dense_layer(50),
    my_dense_layer(10),
    my_dense_layer(10),
    my_dense_layer(10),
    my_dense_layer(10, activation="softmax")
])
```

In [19]:
```
model.compile(loss="sparse_categorical_crossentropy",
             optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
             metrics=["accuracy"])
```

In [20]:
```
history = model.fit(X_train, y_train, epochs=30, validation_data=(X_valid,y_valid))
```

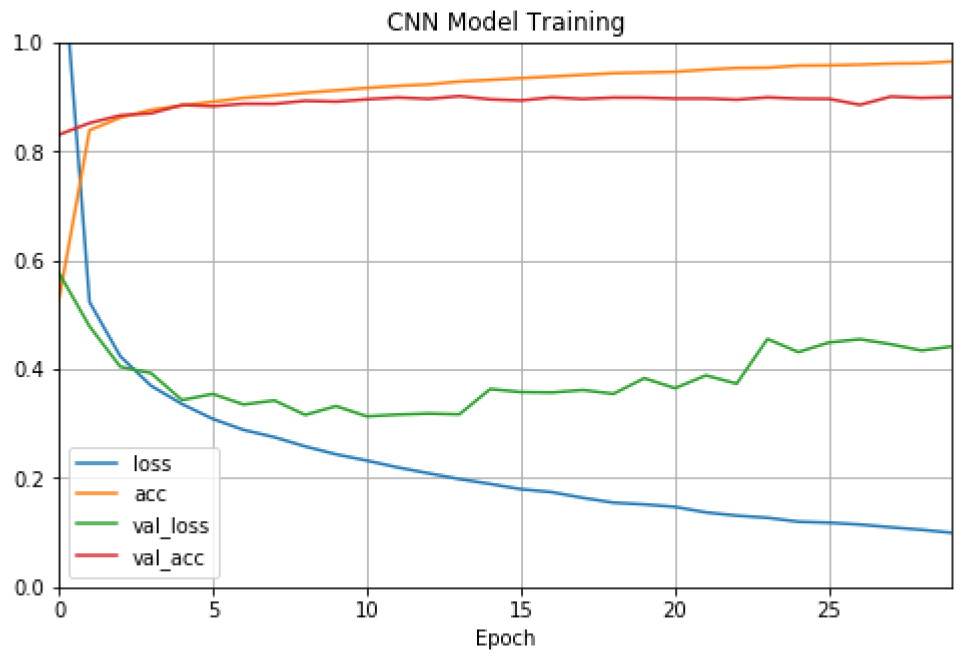
```
Train on 55000 samples, validate on 5000 samples
Epoch 1/30
55000/55000 [==============================] - 43s 787us/sample - loss: 1.2510 - acc: 0.5246 - val_loss: 0.57
78 - val_acc: 0.8306
Epoch 2/30
55000/55000 [==============================] - 37s 676us/sample - loss: 0.5235 - acc: 0.8390 - val_loss: 0.47
88 - val_acc: 0.8522
Epoch 3/30
55000/55000 [==============================] - 33s 595us/sample - loss: 0.4237 - acc: 0.8621 - val_loss: 0.40
45 - val_acc: 0.8662
Epoch 4/30
55000/55000 [==============================] - 34s 613us/sample - loss: 0.3695 - acc: 0.8765 - val_loss: 0.39
27 - val_acc: 0.8700
Epoch 5/30
55000/55000 [==============================] - 35s 628us/sample - loss: 0.3361 - acc: 0.8843 - val_loss: 0.34
33 - val_acc: 0.8854
Epoch 6/30
55000/55000 [==============================] - 34s 614us/sample - loss: 0.3087 - acc: 0.8912 - val_loss: 0.35
46 - val_acc: 0.8830
Epoch 7/30
55000/55000 [==============================] - 32s 588us/sample - loss: 0.2885 - acc: 0.8983 - val_loss: 0.33
51 - val_acc: 0.8876
Epoch 8/30
55000/55000 [==============================] - 32s 581us/sample - loss: 0.2751 - acc: 0.9029 - val_loss: 0.34
26 - val_acc: 0.8874
Epoch 9/30
55000/55000 [==============================] - 32s 577us/sample - loss: 0.2582 - acc: 0.9078 - val_loss: 0.31
61 - val_acc: 0.8930
Epoch 10/30
55000/55000 [==============================] - 34s 610us/sample - loss: 0.2437 - acc: 0.9122 - val_loss: 0.33
22 - val_acc: 0.8916
Epoch 11/30
55000/55000 [==============================] - 34s 618us/sample - loss: 0.2322 - acc: 0.9166 - val_loss: 0.31
34 - val_acc: 0.8958
Epoch 12/30
55000/55000 [==============================] - 34s 622us/sample - loss: 0.2196 - acc: 0.9205 - val_loss: 0.31
67 - val_acc: 0.8992
Epoch 13/30
55000/55000 [==============================] - 34s 623us/sample - loss: 0.2089 - acc: 0.9231 - val_loss: 0.31
86 - val_acc: 0.8966
Epoch 14/30
55000/55000 [==============================] - 35s 633us/sample - loss: 0.1982 - acc: 0.9284 - val_loss: 0.31
71 - val_acc: 0.9016
```

```
Epoch 15/30
55000/55000 [==============================] - 35s 633us/sample - loss: 0.1893 - acc: 0.9314 - val_loss: 0.36
35 - val_acc: 0.8958
Epoch 16/30
55000/55000 [==============================] - 37s 670us/sample - loss: 0.1796 - acc: 0.9346 - val_loss: 0.35
80 - val_acc: 0.8936
Epoch 17/30
55000/55000 [==============================] - 37s 681us/sample - loss: 0.1742 - acc: 0.9376 - val_loss: 0.35
72 - val_acc: 0.8994
Epoch 18/30
55000/55000 [==============================] - 36s 661us/sample - loss: 0.1638 - acc: 0.9407 - val_loss: 0.36
17 - val_acc: 0.8964
Epoch 19/30
55000/55000 [==============================] - 36s 659us/sample - loss: 0.1549 - acc: 0.9438 - val_loss: 0.35
48 - val_acc: 0.8990
Epoch 20/30
55000/55000 [==============================] - 38s 686us/sample - loss: 0.1517 - acc: 0.9450 - val_loss: 0.38
34 - val_acc: 0.8988
Epoch 21/30
55000/55000 [==============================] - 42s 762us/sample - loss: 0.1471 - acc: 0.9463 - val_loss: 0.36
52 - val_acc: 0.8972
Epoch 22/30
55000/55000 [==============================] - 36s 661us/sample - loss: 0.1370 - acc: 0.9502 - val_loss: 0.38
87 - val_acc: 0.8972
Epoch 23/30
55000/55000 [==============================] - 36s 649us/sample - loss: 0.1311 - acc: 0.9533 - val_loss: 0.37
35 - val_acc: 0.8952
Epoch 24/30
55000/55000 [==============================] - 59s 1ms/sample - loss: 0.1271 - acc: 0.9538 - val_loss: 0.4555
- val_acc: 0.8994
Epoch 25/30
55000/55000 [==============================] - 55s 1ms/sample - loss: 0.1199 - acc: 0.9577 - val_loss: 0.4315
- val_acc: 0.8970
Epoch 26/30
55000/55000 [==============================] - 55s 1ms/sample - loss: 0.1181 - acc: 0.9581 - val_loss: 0.4492
- val_acc: 0.8966
Epoch 27/30
55000/55000 [==============================] - 55s 1ms/sample - loss: 0.1147 - acc: 0.9594 - val_loss: 0.4551
- val_acc: 0.8856
Epoch 28/30
55000/55000 [==============================] - 55s 998us/sample - loss: 0.1096 - acc: 0.9614 - val_loss: 0.44
59 - val_acc: 0.9008
Epoch 29/30
```

```
55000/55000 [==============================] - 55s 1ms/sample - loss: 0.1050 - acc: 0.9620 - val_loss: 0.4343
- val_acc: 0.8984
Epoch 30/30
55000/55000 [==============================] - 57s 1ms/sample - loss: 0.0995 - acc: 0.9653 - val_loss: 0.4419
- val_acc: 0.8998
```

In [28]:

```python
pd.DataFrame(history.history).plot(figsize=(8,5))
plt.grid(True)
plt.gca().set_ylim(0,1)
plt.title('CNN Model Training')
plt.xlabel('Epoch')
plt.show()
```



In [23]:
```
y_pred = model.predict_classes(X_train)
conf_train = confusion_matrix(y_train, y_pred)
print(conf_train)
```

```
[[5382    0   32   67    5    0   53    0    4    0]
 [   1 5442    0    0    0    0    0    0    1    0]
 [  13    1 5051   22  345    0   62    0    2    0]
 [   7   13    6 5427   44    1    0    0    1    0]
 [   1    1  117   61 5263    0   64    0    5    0]
 [   0    2    0    2    0 5501    0    2    0    0]
 [ 338    0  161   53  138    0 4816    0    1    0]
 [   0    9    0    0    0    1    0 5470    0    8]
 [   1    1    5    3    0    1    1    0 5498    0]
 [   0    0    0    0    0    6    0   59    0 5429]]
```

In [24]:
```
model.evaluate(X_test,y_test)
```

```
10000/10000 [==============================] - 5s 454us/sample - loss: 0.4803 - acc: 0.8890
```

Out[24]:
```
[0.48029461361020803, 0.889]
```

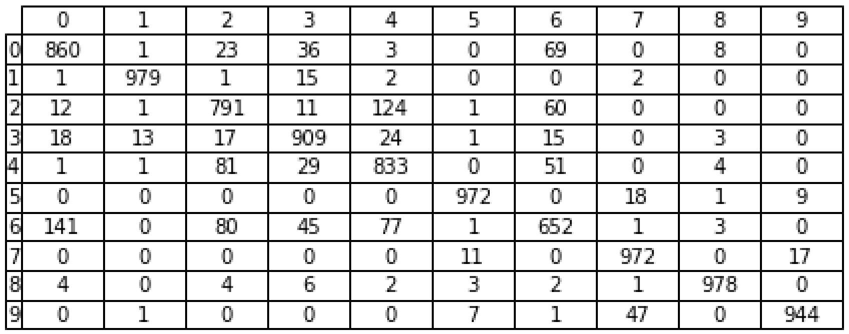
In [25]:
```
y_pred = model.predict_classes(X_test)
conf_test = confusion_matrix(y_test, y_pred)
print(conf_test)
```

```
[[860   1  23  36   3   0  69   0   8   0]
 [  1 979   1  15   2   0   0   2   0   0]
 [ 12   1 791  11 124   1  60   0   0   0]
 [ 18  13  17 909  24   1  15   0   3   0]
 [  1   1  81  29 833   0  51   0   4   0]
 [  0   0   0   0   0 972   0  18   1   9]
 [141   0  80  45  77   1 652   1   3   0]
 [  0   0   0   0   0  11   0 972   0  17]
 [  4   0   4   6   2   3   2   1 978   0]
 [  0   1   0   0   0   7   1  47   0 944]]
```

In [27]:
```
fig, ax = plt.subplots()

# hide axes
#fig.patch.set_visible(False)
ax.axis('off')
ax.axis('tight')

# create table and save to file
df = pd.DataFrame(conf_test)
ax.table(cellText=df.values, rowLabels=np.arange(10), colLabels=np.arange(10), loc='center', cellLoc='center'
)
fig.tight_layout()
plt.savefig('conf_mat_NN.png')
```



	0	1	2	3	4	5	6	7	8	9
0	860	1	23	36	3	0	69	0	8	0
1	1	979	1	15	2	0	0	2	0	0
2	12	1	791	11	124	1	60	0	0	0
3	18	13	17	909	24	1	15	0	3	0
4	1	1	81	29	833	0	51	0	4	0
5	0	0	0	0	0	972	0	18	1	9
6	141	0	80	45	77	1	652	1	3	0
7	0	0	0	0	0	11	0	972	0	17
8	4	0	4	6	2	3	2	1	978	0
9	0	1	0	0	0	7	1	47	0	944

In [ ]:

In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
import math
import pandas as pd
import tensorflow as tf
from sklearn.metrics import confusion_matrix
```

In [2]:
```
fashion_mnist = tf.keras.datasets.fashion_mnist
(X_train_full, y_train_full), (X_test, y_test) = fashion_mnist.load_data()
```

In [3]:
```
X_valid = X_train_full[:5000] / 255.0
X_train = X_train_full[5000:] / 255.0
X_test = X_test / 255.0

y_valid = y_train_full[:5000]
y_train = y_train_full[5000:]

X_train = X_train[..., np.newaxis]
X_valid = X_valid[..., np.newaxis]
X_test = X_test[..., np.newaxis]
```

In [4]:
```
from functools import partial

my_dense_layer = partial(tf.keras.layers.Dense, activation="tanh", kernel_regularizer=tf.keras.regularizers.l
2(0.001))
my_conv_layer = partial(tf.keras.layers.Conv2D, activation="tanh", padding="valid")

model = tf.keras.models.Sequential([
    my_conv_layer(6,5,padding="same",input_shape=[28,28,1]),
    tf.keras.layers.AveragePooling2D(2),
    my_conv_layer(36,5),
    tf.keras.layers.AveragePooling2D(2),
    my_conv_layer(108,5),
    tf.keras.layers.Flatten(),
    my_dense_layer(84),
    my_dense_layer(42),
    my_dense_layer(21),
    my_dense_layer(10),
    my_dense_layer(10, activation="softmax")
])
```

```
WARNING:tensorflow:From C:\Users\whtho\Documents\Python\lib\site-packages\tensorflow\python\ops\init_ops.py:1
251: calling VarianceScaling.__init__ (from tensorflow.python.ops.init_ops) with dtype is deprecated and will
be removed in a future version.
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to the constructor
```

In [5]:
```
model.compile(loss="sparse_categorical_crossentropy",
             optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
             metrics=["accuracy"])
```

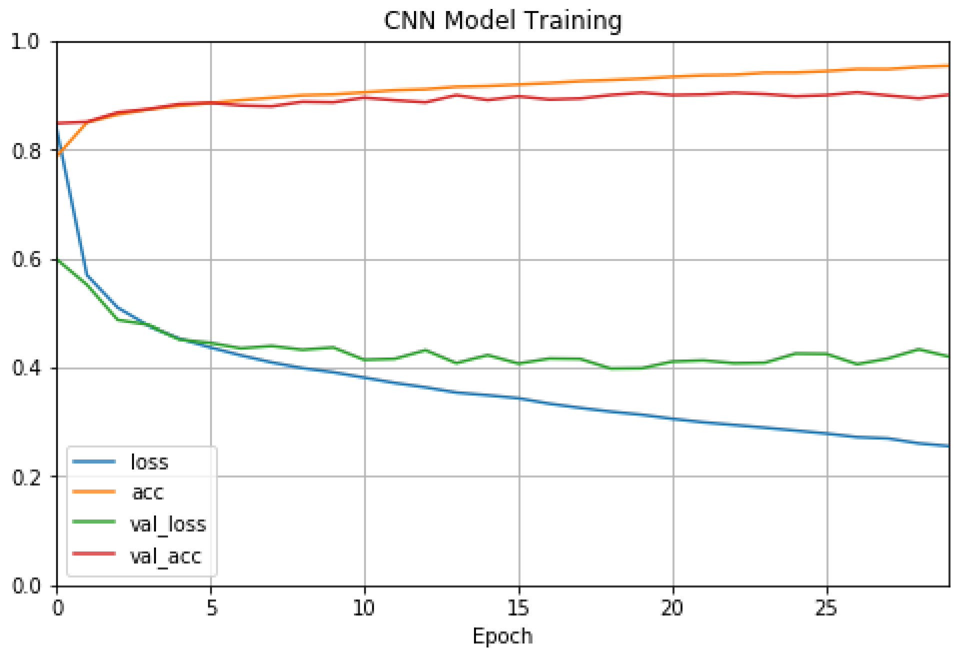
In [6]:
```
history = model.fit(X_train, y_train, epochs=30, validation_data=(X_valid,y_valid))
```

```
Train on 55000 samples, validate on 5000 samples
Epoch 1/30
55000/55000 [==============================] - 67s 1ms/sample - loss: 0.8425 - acc: 0.7868 - val_loss: 0.5994
- val_acc: 0.8486
Epoch 2/30
55000/55000 [==============================] - 76s 1ms/sample - loss: 0.5695 - acc: 0.8500 - val_loss: 0.5517
- val_acc: 0.8512
Epoch 3/30
55000/55000 [==============================] - 76s 1ms/sample - loss: 0.5095 - acc: 0.8646 - val_loss: 0.4870
- val_acc: 0.8686
Epoch 4/30
55000/55000 [==============================] - 75s 1ms/sample - loss: 0.4763 - acc: 0.8740 - val_loss: 0.4780
- val_acc: 0.8748
Epoch 5/30
55000/55000 [==============================] - 70s 1ms/sample - loss: 0.4536 - acc: 0.8808 - val_loss: 0.4519
- val_acc: 0.8840
Epoch 6/30
55000/55000 [==============================] - 75s 1ms/sample - loss: 0.4374 - acc: 0.8858 - val_loss: 0.4458
- val_acc: 0.8864
Epoch 7/30
55000/55000 [==============================] - 74s 1ms/sample - loss: 0.4227 - acc: 0.8911 - val_loss: 0.4356
- val_acc: 0.8814
Epoch 8/30
55000/55000 [==============================] - 76s 1ms/sample - loss: 0.4097 - acc: 0.8956 - val_loss: 0.4400
- val_acc: 0.8798
Epoch 9/30
55000/55000 [==============================] - 74s 1ms/sample - loss: 0.3992 - acc: 0.9001 - val_loss: 0.4330
- val_acc: 0.8882
Epoch 10/30
55000/55000 [==============================] - 77s 1ms/sample - loss: 0.3916 - acc: 0.9017 - val_loss: 0.4372
- val_acc: 0.8872
Epoch 11/30
55000/55000 [==============================] - 80s 1ms/sample - loss: 0.3818 - acc: 0.9055 - val_loss: 0.4147
- val_acc: 0.8958
Epoch 12/30
55000/55000 [==============================] - 78s 1ms/sample - loss: 0.3720 - acc: 0.9094 - val_loss: 0.4163
- val_acc: 0.8910
Epoch 13/30
55000/55000 [==============================] - 75s 1ms/sample - loss: 0.3638 - acc: 0.9113 - val_loss: 0.4323
- val_acc: 0.8872
Epoch 14/30
55000/55000 [==============================] - 77s 1ms/sample - loss: 0.3543 - acc: 0.9157 - val_loss: 0.4084
- val_acc: 0.9004
```

```
Epoch 15/30
55000/55000 [==============================] - 76s 1ms/sample - loss: 0.3493 - acc: 0.9171 - val_loss: 0.4232
- val_acc: 0.8914
Epoch 16/30
55000/55000 [==============================] - 64s 1ms/sample - loss: 0.3438 - acc: 0.9199 - val_loss: 0.4076
- val_acc: 0.8982
Epoch 17/30
55000/55000 [==============================] - 75s 1ms/sample - loss: 0.3338 - acc: 0.9226 - val_loss: 0.4169
- val_acc: 0.8924
Epoch 18/30
55000/55000 [==============================] - 82s 1ms/sample - loss: 0.3261 - acc: 0.9261 - val_loss: 0.4163
- val_acc: 0.8942
Epoch 19/30
55000/55000 [==============================] - 67s 1ms/sample - loss: 0.3191 - acc: 0.9282 - val_loss: 0.3984
- val_acc: 0.9006
Epoch 20/30
55000/55000 [==============================] - 26s 480us/sample - loss: 0.3135 - acc: 0.9307 - val_loss: 0.39
90 - val_acc: 0.9050
Epoch 21/30
55000/55000 [==============================] - 27s 490us/sample - loss: 0.3060 - acc: 0.9341 - val_loss: 0.41
14 - val_acc: 0.9004
Epoch 22/30
55000/55000 [==============================] - 26s 479us/sample - loss: 0.2996 - acc: 0.9368 - val_loss: 0.41
35 - val_acc: 0.9016
Epoch 23/30
55000/55000 [==============================] - 26s 477us/sample - loss: 0.2946 - acc: 0.9378 - val_loss: 0.40
83 - val_acc: 0.9048
Epoch 24/30
55000/55000 [==============================] - 26s 480us/sample - loss: 0.2896 - acc: 0.9416 - val_loss: 0.40
91 - val_acc: 0.9024TA: 0s - loss: 0.2899 - acc: 0
Epoch 25/30
55000/55000 [==============================] - 27s 491us/sample - loss: 0.2843 - acc: 0.9420 - val_loss: 0.42
60 - val_acc: 0.8982
Epoch 26/30
55000/55000 [==============================] - 27s 493us/sample - loss: 0.2789 - acc: 0.9445 - val_loss: 0.42
54 - val_acc: 0.9004
Epoch 27/30
55000/55000 [==============================] - 27s 492us/sample - loss: 0.2722 - acc: 0.9483 - val_loss: 0.40
65 - val_acc: 0.9056
Epoch 28/30
55000/55000 [==============================] - 27s 492us/sample - loss: 0.2699 - acc: 0.9482 - val_loss: 0.41
69 - val_acc: 0.8998
Epoch 29/30
```

```
55000/55000 [==============================] - 27s 488us/sample - loss: 0.2609 - acc: 0.9522 - val_loss: 0.43
39 - val_acc: 0.8944
Epoch 30/30
55000/55000 [==============================] - 27s 491us/sample - loss: 0.2557 - acc: 0.9545 - val_loss: 0.42
02 - val_acc: 0.9012
```

In [13]:
```
pd.DataFrame(history.history).plot(figsize=(8,5))
plt.grid(True)
plt.gca().set_ylim(0,1)
plt.title('CNN Model Training')
plt.xlabel('Epoch')
plt.show()
```



In [8]:
```
y_pred = model.predict_classes(X_train)
conf_train = confusion_matrix(y_train, y_pred)
print(conf_train)
```

```
[[5193    2   11   58    0    1  271    0    7    0]
 [   1 5416    1   26    0    0    0    0    0    0]
 [  28    1 5208   15  159    0   84    0    1    0]
 [  14    4   11 5386   61    0   23    0    0    0]
 [   1    1  193   83 5109    0  124    0    1    0]
 [   0    0    0    0    0 5496    0   10    0    1]
 [ 293    1  117   83  103    0 4907    0    3    0]
 [   0    0    0    0    0   47    0 5401    1   39]
 [   7    0    5    3    2    0    9    0 5483    1]
 [   0    0    0    0    0    7    0  119    1 5367]]
```

In [9]:
```
model.evaluate(X_test,y_test)
```

```
10000/10000 [==============================] - 2s 181us/sample - loss: 0.4464 - acc: 0.8935
```

Out[9]:
```
[0.44643619248867034, 0.8935]
```

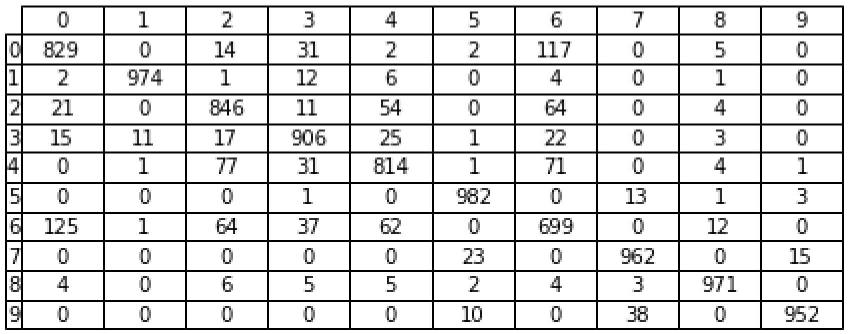
In [10]:
```
y_pred = model.predict_classes(X_test)
conf_test = confusion_matrix(y_test, y_pred)
print(conf_test)
```

```
[[829   0  14  31   2   2 117   0   5   0]
 [  2 974   1  12   6   0   4   0   1   0]
 [ 21   0 846  11  54   0  64   0   4   0]
 [ 15  11  17 906  25   1  22   0   3   0]
 [  0   1  77  31 814   1  71   0   4   1]
 [  0   0   0   1   0 982   0  13   1   3]
 [125   1  64  37  62   0 699   0  12   0]
 [  0   0   0   0   0  23   0 962   0  15]
 [  4   0   6   5   5   2   4   3 971   0]
 [  0   0   0   0   0  10   0  38   0 952]]
```

In [20]:
```
fig, ax = plt.subplots()

# hide axes
#fig.patch.set_visible(False)
ax.axis('off')
ax.axis('tight')

# create table and save to file
df = pd.DataFrame(conf_test)
ax.table(cellText=df.values, rowLabels=np.arange(10), colLabels=np.arange(10), loc='center', cellLoc='center'
)
fig.tight_layout()
plt.savefig('CNN_conf_mat.png')
```



	0	1	2	3	4	5	6	7	8	9
0	829	0	14	31	2	2	117	0	5	0
1	2	974	1	12	6	0	4	0	1	0
2	21	0	846	11	54	0	64	0	4	0
3	15	11	17	906	25	1	22	0	3	0
4	0	1	77	31	814	1	71	0	4	1
5	0	0	0	1	0	982	0	13	1	3
6	125	1	64	37	62	0	699	0	12	0
7	0	0	0	0	0	23	0	962	0	15
8	4	0	6	5	5	2	4	3	971	0
9	0	0	0	0	0	10	0	38	0	952

In [ ]: