Tam Phi
ECE 612 – Project 3

Introduction

In this project, student will develop a CNN model to classify a fashion MNIST dataset.

Design and Implementation

Data Processing

Data (X_train,y_train) will split into 2 sets:

- Train set:
  - Training
  - Validation
- Test set

```
14 y_train = pd.read_pickle('project3trainlabel.pkl')
15 X_train = pd.read_pickle('project3trainset.pkl')
```

Each set will then be normalized and added a new channel to the set (line 74 – 76)

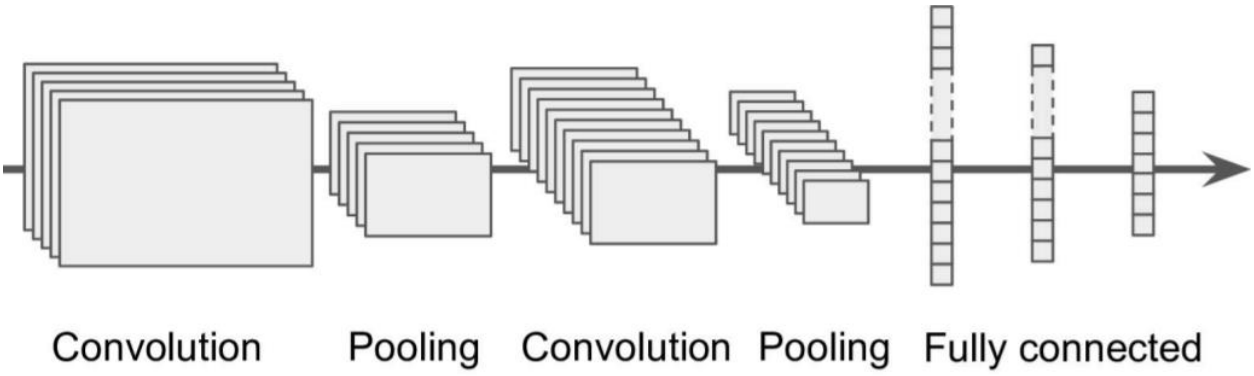
```
51 from sklearn.model_selection import train_test_split
52 X_train_set, X_test_set = train_test_split(X_train, test_size=0.1, random_state=42)
53 y_train_set, y_test_set = train_test_split(y_train, test_size=0.1, random_state=42)
54
55 X_train_set, X_valid_set= X_train_set[:-5000], X_train_set[-5000:]
56 y_train_set, y_valid_set = y_train_set[:-5000], y_train_set[-5000:]
57 X_mean = X_train_set.mean(axis=0, keepdims=True)
58 X_std = X_train_set.std(axis=0, keepdims=True) + 1e-7
59 X_train_set = (X_train_set - X_mean) / X_std
60 X_valid_set = (X_valid_set - X_mean) / X_std
61 X_test_set = (X_test_set - X_mean) / X_std
```

```
74 X_train_set = X_train_set[..., np.newaxis]
75 X_valid_set = X_valid_set[..., np.newaxis]
76 X_test_set = X_test_set[..., np.newaxis]
```

Grid Search and Hyperparameters Tuning

The model used consists of different layers of Convolution, pooling, and dense layer (fully connected)

Tam Phi
ECE 612 – Project 3



Because the total parameters must be under 50,000, the filters and units size will be restricted to 36 at maximum.

```
def create_model(activation = 'relu',
                 optimizer = "nadam",
                 dropout_rate = 0.2,
                 kernel =3,
                 ):
    DefaultConv2D = partial(keras.layers.Conv2D,
    kernel_size=kernel, activation=activation, padding="SAME")
    model_grid = keras.models.Sequential([
        DefaultConv2D(filters=28, kernel_size=7, input_shape=[28, 28, 1]),
        keras.layers.MaxPooling2D(pool_size=3),
        DefaultConv2D(filters=32),
        DefaultConv2D(filters=32),
        keras.layers.MaxPooling2D(pool_size=2),
        DefaultConv2D(filters=36),
        DefaultConv2D(filters=36),
        keras.layers.MaxPooling2D(pool_size=2),
        keras.layers.Flatten(),
        keras.layers.Dense(units=36, activation= activation),
        keras.layers.Dropout(dropout_rate),
        keras.layers.Dense(units=32, activation= activation),
        keras.layers.Dropout(dropout_rate),
        keras.layers.Dense(units=10, activation= 'softmax'),
])
    model_grid.compile(loss="sparse_categorical_crossentropy", optimizer=optimizer, metrics=["accuracy"])
    return model_grid
```

The function to create the model (create_model) has the following modifiable parameters:

- activation
- dropout_rate
- optimizers
- epochs

```
activation =  ['relu', 'sigmoid'] # softmax, softplus, softsign
dropout_rate = [ 0.2, 0.3, 0.4]
optimizers = ['Adadelta', 'Nadam']
epochs = [20,30] # add 50, 100, 150 etc
param_grid = dict(activation=activation,optimizer = optimizers,dropout_rate = dropout_rate, epochs=epochs)
```

The results for the best parameters are:

```
In [166]: print(grid_result.best_params_)
     ...: print(grid_result.best_score_)
{'activation': 'relu', 'dropout_rate': 0.2, 'epochs': 20, 'optimizer': 'Adadelta'}
0.8564719080924987
```

Model Selection

While the grid search reports that the best parameters are:

- activation = ‘relu’
- dropout_rate = 0.2
- epochs = 20
- optimizer = ‘Adadelta’

The low value for dropout rate could leads to over fitting. Therefore, with the first dense layer has lower value of dropout (0.2) while the next one has higher value (0.4).

```
DefaultConv2D = partial(keras.layers.Conv2D,
kernel_size=3, activation='relu', padding="SAME")
model = keras.models.Sequential([
        DefaultConv2D(filters=28, kernel_size=7, input_shape=[28, 28, 1]),
        keras.layers.MaxPooling2D(pool_size=3),
        DefaultConv2D(filters=32),
        DefaultConv2D(filters=32),
        keras.layers.MaxPooling2D(pool_size=2),
        DefaultConv2D(filters=36),
        DefaultConv2D(filters=36),
        keras.layers.MaxPooling2D(pool_size=2),
        keras.layers.Flatten(),
        keras.layers.Dense(units=36, activation='relu'),
        keras.layers.Dropout(0.2),
        keras.layers.Dense(units=32, activation='relu'),
        keras.layers.Dropout(0.4),
        keras.layers.Dense(units=10, activation='softmax'),
])
```

Layer (type)	Output Shape	Param #
conv2d_1045 (Conv2D)	(None, 28, 28, 28)	1400
max_pooling2d_627 (MaxPoolin	(None, 9, 9, 28)	0
conv2d_1046 (Conv2D)	(None, 9, 9, 32)	8096
conv2d_1047 (Conv2D)	(None, 9, 9, 32)	9248
max_pooling2d_628 (MaxPoolin	(None, 4, 4, 32)	0
conv2d_1048 (Conv2D)	(None, 4, 4, 36)	10404
conv2d_1049 (Conv2D)	(None, 4, 4, 36)	11700
max_pooling2d_629 (MaxPoolin	(None, 2, 2, 36)	0
flatten_209 (Flatten)	(None, 144)	0
dense_627 (Dense)	(None, 36)	5220
dropout_418 (Dropout)	(None, 36)	0
dense_628 (Dense)	(None, 32)	1184
dropout_419 (Dropout)	(None, 32)	0
dense_629 (Dense)	(None, 10)	330

Total params: 47,582
Trainable params: 47,582
Non-trainable params: 0

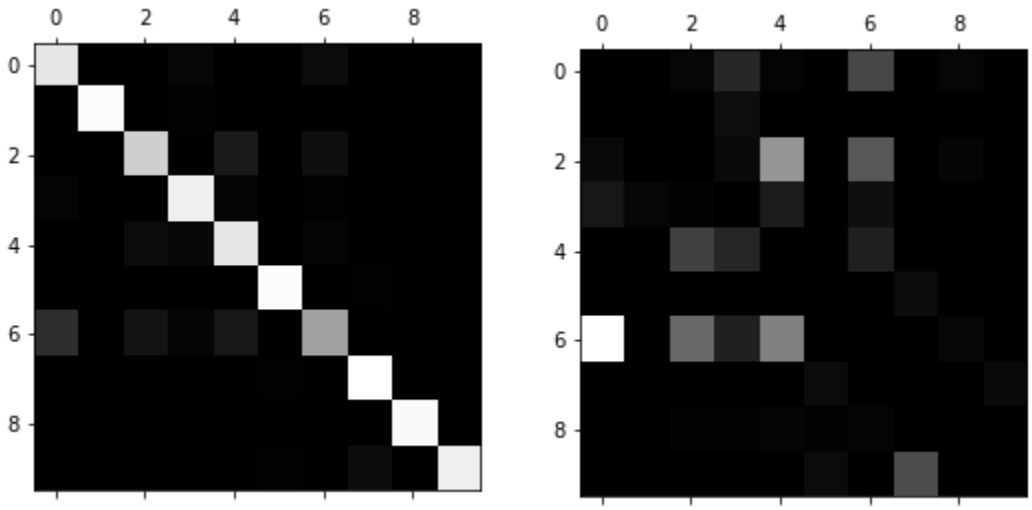
```
Epoch 17/20
44500/44500 [==============================] - 32s 715us/sample - loss: 0.3340 - acc: 0.8846 - val_loss: 0.4475 - val_acc:
0.8476
Epoch 18/20
44500/44500 [==============================] - 34s 761us/sample - loss: 0.3346 - acc: 0.8873 - val_loss: 0.4742 - val_acc:
0.8528
Epoch 19/20
44500/44500 [==============================] - 32s 726us/sample - loss: 0.3329 - acc: 0.8855 - val_loss: 0.4881 - val_acc:
0.8548
Epoch 20/20
44500/44500 [==============================] - 32s 723us/sample - loss: 0.3320 - acc: 0.8851 - val_loss: 0.5511 - val_acc:
0.8572
```

The result is a high accuracy of 88% (full run on 20 epochs can be found at the end of this report)

Result and Discussion

The confusion matrix (left) for this model seems to perform reasonably well. Class #6 appears to be darker than other classes, so the model doesn't classify it as well.

The error matrix (right) indicates that the classifier often mistakes Class #6 as Class #0.



The precision, recall, and f1 score for this CNN classifier is high. The f1 scores are mostly in the 0.83 - 0.98 range apart from Class #6, which is also reflected in the error and confusion matrix.

N	Precision	Recall	F1-Score
0	0.9084523000226603	0.813019671466234	0.8580907534246575
1	0.9873817034700315	0.9936507936507937	0.990506329113924
2	0.8141054529067148	0.8614687649022413	0.8371177015755329
3	0.9438787055894999	0.9049685398134085	0.9240141781125388
4	0.9008968609865471	0.8007174172977282	0.8478581979320532
5	0.9891989198919892	0.981031019861638	0.9850980392156863
6	0.6268124024091011	0.8090987618773394	0.7063851181498242
7	0.9840954274353877	0.938685208596713	0.9608540925266904
8	0.9844173441734417	0.98330701556508	0.9838618666064779
9	0.9375280898876405	0.9921521997621878	0.9640670132871172

Full results:

Train on 44500 samples, validate on 5000 samples

Epoch 1/20

44500/44500 [==============================] - 51s 1ms/sample - loss: 0.8932 - acc: 0.6767 - val_loss: 0.5447 - val_acc: 0.7952

Epoch 2/20

44500/44500 [==============================] - 31s 700us/sample - loss: 0.5970 - acc: 0.7882 - val_loss: 0.4725 - val_acc: 0.8292

Epoch 3/20

44500/44500 [==============================] - 31s 688us/sample - loss: 0.5311 - acc: 0.8166 - val_loss: 0.4376 - val_acc: 0.8426

Epoch 4/20

44500/44500 [==============================] - 30s 675us/sample - loss: 0.4869 - acc: 0.8315 - val_loss: 0.4406 - val_acc: 0.8390

Epoch 5/20

44500/44500 [==============================] - 29s 652us/sample - loss: 0.4591 - acc: 0.8415 - val_loss: 0.4191 - val_acc: 0.8532

Epoch 6/20

44500/44500 [==============================] - 29s 656us/sample - loss: 0.4375 - acc: 0.8509 - val_loss: 0.4307 - val_acc: 0.8536

Epoch 7/20

44500/44500 [==============================] - 30s 674us/sample - loss: 0.4226 - acc: 0.8565 - val_loss: 0.4296 - val_acc: 0.8556

Epoch 8/20

44500/44500 [==============================] - 30s 666us/sample - loss: 0.4041 - acc: 0.8613 - val_loss: 0.4396 - val_acc: 0.8470

Epoch 9/20

44500/44500 [==============================] - 31s 688us/sample - loss: 0.3947 - acc: 0.8662 - val_loss: 0.4445 - val_acc: 0.8512

Epoch 10/20

44500/44500 [==============================] - 30s 674us/sample - loss: 0.3827 - acc: 0.8679 - val_loss: 0.4164 - val_acc: 0.8588

Epoch 11/20

44500/44500 [==============================] - 29s 661us/sample - loss: 0.3748 - acc: 0.8730 - val_loss: 0.4443 - val_acc: 0.8576

Epoch 12/20

44500/44500 [==============================] - 29s 660us/sample - loss: 0.3605 - acc: 0.8753 - val_loss: 0.4379 - val_acc: 0.8544

Epoch 13/20

44500/44500 [==============================] - 31s 694us/sample - loss: 0.3570 - acc: 0.8776 - val_loss: 0.4515 - val_acc: 0.8568

Epoch 14/20

44500/44500 [==============================] - 30s 666us/sample - loss: 0.3500 - acc: 0.8784 - val_loss: 0.5069 - val_acc: 0.8596

Epoch 15/20

44500/44500 [==============================] - 32s 725us/sample - loss: 0.3445 - acc: 0.8812 - val_loss: 0.4809 - val_acc: 0.8548

Epoch 16/20

44500/44500 [==============================] - 32s 709us/sample - loss: 0.3440 - acc: 0.8826 - val_loss: 0.4828 - val_acc: 0.8562

Epoch 17/20

44500/44500 [==============================] - 32s 715us/sample - loss: 0.3340 - acc: 0.8846 - val_loss: 0.4475 - val_acc: 0.8476

Epoch 18/20

44500/44500 [==============================] - 34s 761us/sample - loss: 0.3346 - acc: 0.8873 - val_loss: 0.4742 - val_acc: 0.8528

Epoch 19/20

44500/44500 [==============================] - 32s 726us/sample - loss: 0.3329 - acc: 0.8855 - val_loss: 0.4881 - val_acc: 0.8548

Epoch 20/20

44500/44500 [==============================] - 32s 723us/sample - loss: 0.3320 - acc: 0.8851 - val_loss: 0.5511 - val_acc: 0.8572

5500/5500 [==============================] - 3s 483us/sample - loss: 0.6219 - acc: 0.85291s - loss: 0.6221 - acc: 0.8566