FashionMLP

August 29, 2020

1 Introduction

Laboratory 3 focuses on the ability for the programmer to categorize images corresponding to different fashion items. The items ilist ncludes distinct things like shirts, sneakers, and bags. On the other hand, the list also includes similar pairs like shirts and tops, and sneaker and sandal which will really test the algorithms ability to separate them. There are many approaches for a supervised algorithm. The problem will test not only the accuracy, but its efficiency. Models like linear classification will be extremely faster relative to neural network models. Adversely, we will need to figure out if speed will sacrifice accuracy significantly.

2 Imports

[246]:
```
import os
import struct
import numpy as np
import pandas as pd
import seaborn as sns
import time
import matplotlib.pyplot as plt
import sys
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, BatchNormalization
from keras.layers import Activation, Dropout, Flatten, Dense
from keras.utils import to_categorical
from sklearn.linear_model import SGDClassifier

%matplotlib inline
```

3 Set Path

[20]:
```
pwd
```

[20]:
```
'/Users/erm1000255241/Library/Mobile Documents/com~apple~CloudDocs/Documents/Syr
acuseUniversity/4th_Quarter/IST718/Lab3'
```

[19]:
```
path = '/Users/erm1000255241/Library/Mobile Documents/com~apple~CloudDocs/
 ↪Documents/SyracuseUniversity/4th_Quarter/IST718/Lab3/'
```

4 Read in Data

To read in the data, we import the mnist_ready.py file that is provided in the github page.

[66]:
```
import mnist_reader
X_train, y_train = mnist_reader.load_mnist('data/', kind='train')
X_test, y_test = mnist_reader.load_mnist('data/', kind='t10k')
```

[67]:
```
print('Rows: %d, columns: %d' % (X_train.shape[0], X_train.shape[1]))
```

```
Rows: 60000, columns: 784
```

[68]:
```
print('Rows: %d, columns: %d' % (X_test.shape[0], X_test.shape[1]))
```

```
Rows: 10000, columns: 784
```

4.1 Create Dictionary for Labels

For an easier visualization, we create a dictionary that will easily allow us to convert the number label to an actual category.

[219]:
```
labelD = {
0:'T-shirt/top',
1:'Trouser',
2:'Pullover',
3:'Dress',
4:'Coat',
5:'Sandal',
6:'Shirt',
7:'Sneaker',
8:'Bag',
9:'Ankle boot',
}
```

5 Visualize Dataset

[71]:
```
np.unique(y_train)
```

[71]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], dtype=uint8)
```

[87]:
```
# VISUALIZE DIFFERENT VARIATIONS OF Each Category
nRows = 4
numCats = len(np.unique(y_train))
```

```
fig, ax = plt.subplots(nrows=nRows, ncols=numCats, sharex=True,␣
 ↪sharey=True,figsize=(8, 6))
ax = ax.flatten()
for i in range(numCats * nRows):
    img = X_train[y_train == i%numCats][i//numCats].reshape(28, 28)
    ax[i].imshow(img, cmap='Greys', interpolation='nearest')

ax[0].set_xticks([])
ax[0].set_yticks([])
plt.tight_layout()
# plt.savefig('./figures/mnist_9.png', dpi=300)
plt.show()
```

From a simple visual, we can already start to see some problems that might arise later on while modeling. Tops and Shirts are likely to overlap and cause many incorrect predictins. We need to watch out for these similarities in the models' results.

6 Preprocess and Analyze Dataset

6.1 Get Counts

6.1.1 Training

[332]:
```
countsTrain = np.unique(y_train, return_counts=True)
```

[333]:
```
pd.DataFrame(zip(countsTrain[0], countsTrain[1]), columns=['label', 'counts'])
```

[333]:

	label	counts
0	0	6000
1	1	6000
2	2	6000
3	3	6000
4	4	6000
5	5	6000
6	6	6000
7	7	6000
8	8	6000
9	9	6000

When creating a supervised model, we want a balanced dataset. The dataset provided to us does follow this guidline as each category has 6000 examples.

6.1.2 Test Data

[334]:
```
countsTest = np.unique(y_test, return_counts=True)
```

[335]:
```
pd.DataFrame(zip(countsTest[0], countsTest[1]), columns=['label', 'counts'])
```

[335]:

	label	counts
0	0	1000
1	1	1000
2	2	1000
3	3	1000
4	4	1000
5	5	1000
6	6	1000
7	7	1000
8	8	1000
9	9	1000

The same goes for the testing set. We expect to get a much cleaner understanding of our models accuracy since there is an equal amount of test pictures per category.

7 Create Models

7.1 Create Function to get prediction accuracy

For simplicity, we create a function that will allow us to get three different dataframes from each prediction.

[237]:
```
def predAccuracy(actual, pred):
    acc_df = pd.DataFrame(zip(actual, pred), columns=['ac_label', 'pred_label'])
    acc_df['actual'] = [labelD[x] for x in acc_df['ac_label'].values]
    acc_df['pred'] = [labelD[x] for x in acc_df['pred_label'].values]
    acc_df['Correct'] = acc_df['pred'] == acc_df['actual']

    acc_counts = acc_df[['actual', 'pred']].groupby(['actual', 'pred']).size().
 ↪unstack(fill_value=0)

    acc_perc = acc_counts.div(acc_counts.sum(axis=1), axis=0)*100

    return acc_df, acc_counts, acc_perc
```

7.2 Simple Linear Classification

To start off predicting, we want to set up a baseline. Because of Occam's Razor, we dont want to create a model that is too complex and sacrifices its accuracy. Therefore, once we create more complex model's, we expect its accuracy to increase. Otherwise, we should stick with the simplest method. In other words, the model's complexity has to match the data's complexity.

[274]:
```
# create the linear model classifier
clf = SGDClassifier()
# fit (train) the classifier
clf.fit(X_train, y_train)
```

[274]:
```
SGDClassifier(alpha=0.0001, average=False, class_weight=None,
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='hinge',
              max_iter=1000, n_iter_no_change=5, n_jobs=None, penalty='l2',
              power_t=0.5, random_state=None, shuffle=True, tol=0.001,
              validation_fraction=0.1, verbose=0, warm_start=False)
```

Training

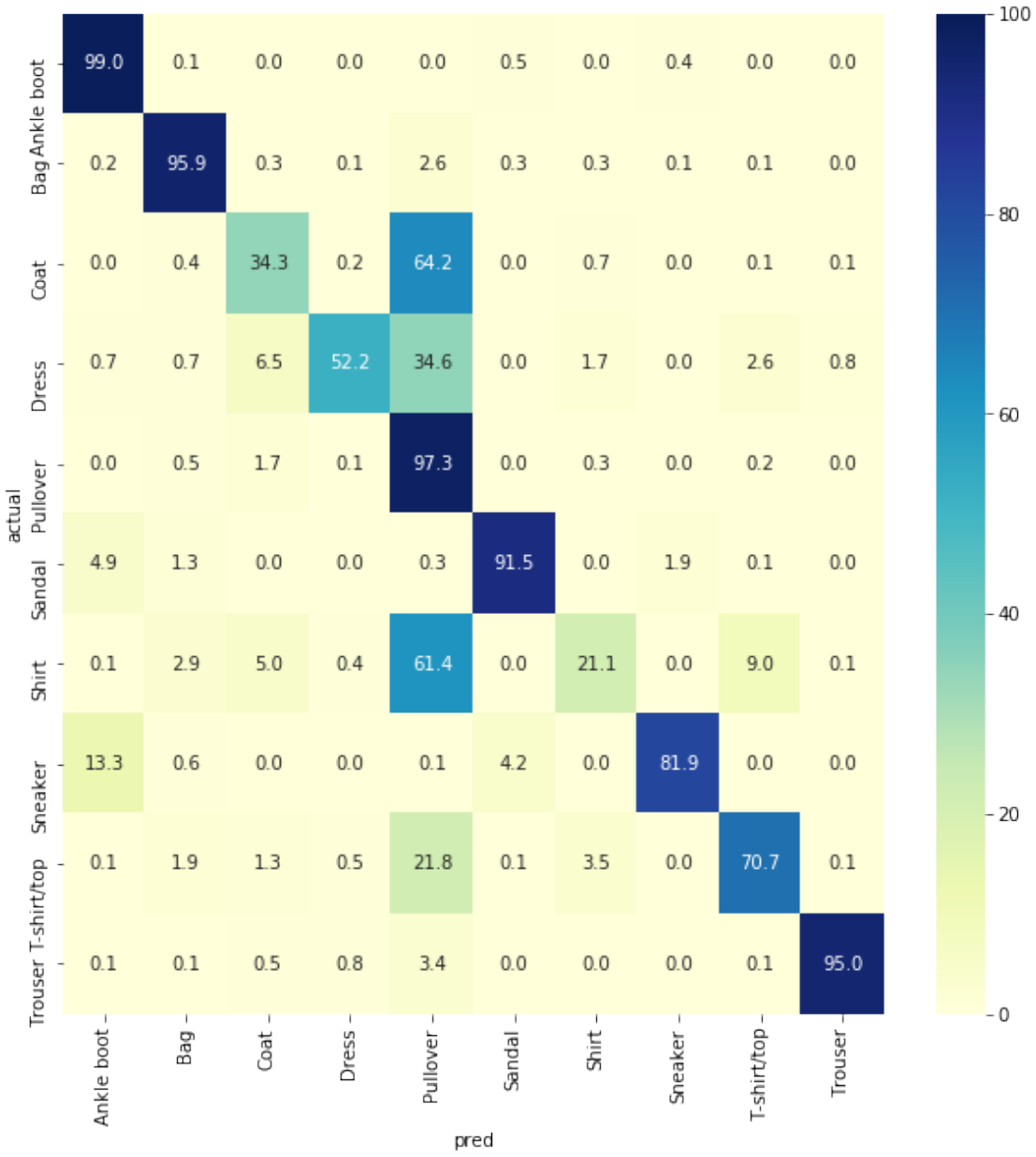
[275]:
```
y_train_pred = clf.predict(X_train)

if sys.version_info < (3, 0):
    acc = (np.sum(y_train == y_train_pred, axis=0)).astype('float') / X_train.
 ↪shape[0]
else:
    acc = np.sum(y_train == y_train_pred, axis=0) / X_train.shape[0]
```

```
print('Training accuracy: %.2f%%' % (acc * 100))
```

```
Training accuracy: 73.89%
```

[276]:
```
acc_df_train_clf, acc_counts_train_clf, acc_perc_train_clf =␣
↪predAccuracy(y_train, y_train_pred)
```

[277]:
```
plt.figure(figsize=(10,10))
ax = sns.heatmap(acc_perc_train_clf, vmin=0, vmax=100, annot=True, fmt=".1f",␣
↪cmap="YlGnBu")
```



The training set's prediction from the Linear Classification model is already showing the similar category pairs difficulty separating them. Coats have a higher probability to be classified as Pullovers than actual Coats. The same goes for Shirts. It seems that anything that resembles a pullover will most likely be classified as a pullover.

Test

[278]:
```
y_test_pred = clf.predict(X_test)

if sys.version_info < (3, 0):
    acc = (np.sum(y_test == y_test_pred, axis=0)).astype('float') / X_test.
 ↪shape[0]
else:
    acc = np.sum(y_test == y_test_pred, axis=0) / X_test.shape[0]

print('Test accuracy: %.2f%%' % (acc * 100))
```

```
Test accuracy: 71.73%
```

[280]:
```
acc_df_test_clf, acc_counts_test_clf, acc_perc_test_clf = predAccuracy(y_test,␣
 ↪y_test_pred)
```

[336]:
```
# plt.figure(figsize=(10,10))
# ax = sns.heatmap(acc_perc_test_clf, vmin=0, vmax=100, annot=True, fmt=".1f",␣
 ↪cmap="YlGnBu")
```

The test accuracy of the linear classification model is 71.73% which is a good start, but it is not close to the accuracy that we want.

7.3 Neural Network

The next model we want to chose is a Neural Network. The Neural Network shown in class is a good example of how to create one from scratch. On the other hand, it can be too simple compared to a library that has been pre-defined for problems just like this.

7.3.1 Create Model Class

[88]:
```
# FIRST PASS WITH MLP

import numpy as np
from scipy.special import expit
import sys


class NeuralNetMLP(object):
```

```
        return self
```

7.3.2 Compile and Fit Model

[89]:
```
nn = NeuralNetMLP(n_output=10,
                  n_features=X_train.shape[1],
                  n_hidden=50,
                  l2=0.1,
                  l1=0.0,
                  epochs=1000,
                  eta=0.001,
                  alpha=0.001,
                  decrease_const=0.00001,
                  minibatches=50,
                  shuffle=True,
                  random_state=1)
```

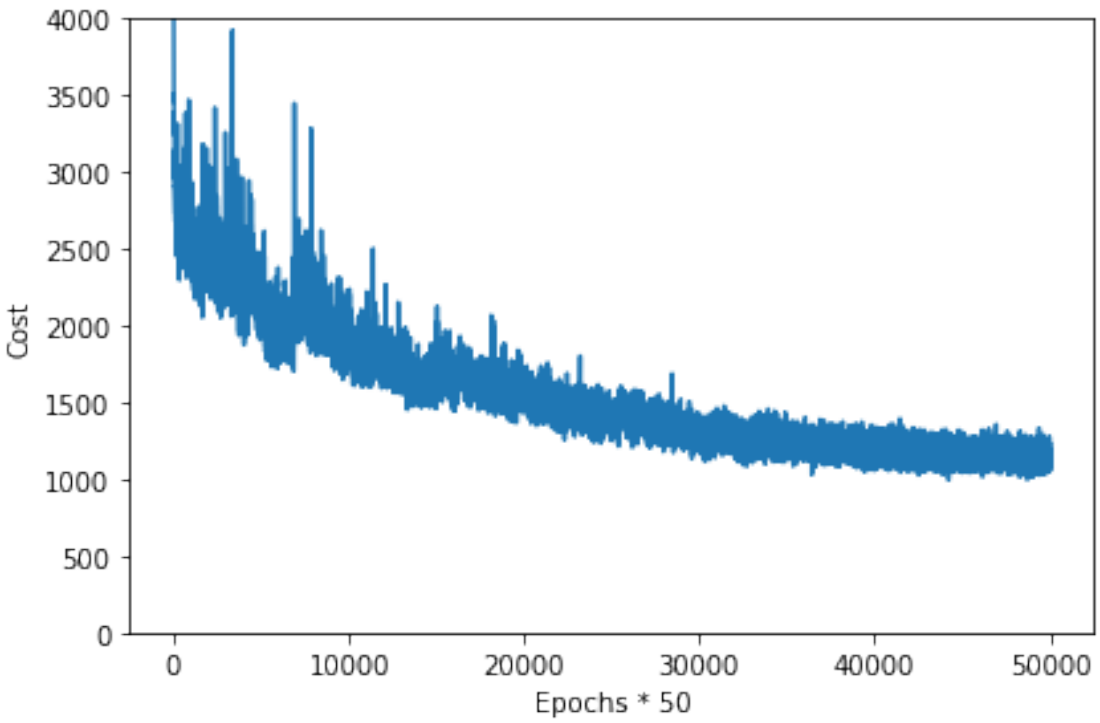
[90]:
```
start = time.time()

nn.fit(X_train, y_train, print_progress=True)

end = time.time()
final_time = end-start
print(final_time)
```

```
Epoch: 1000/1000
```

```
309.833340883255
```

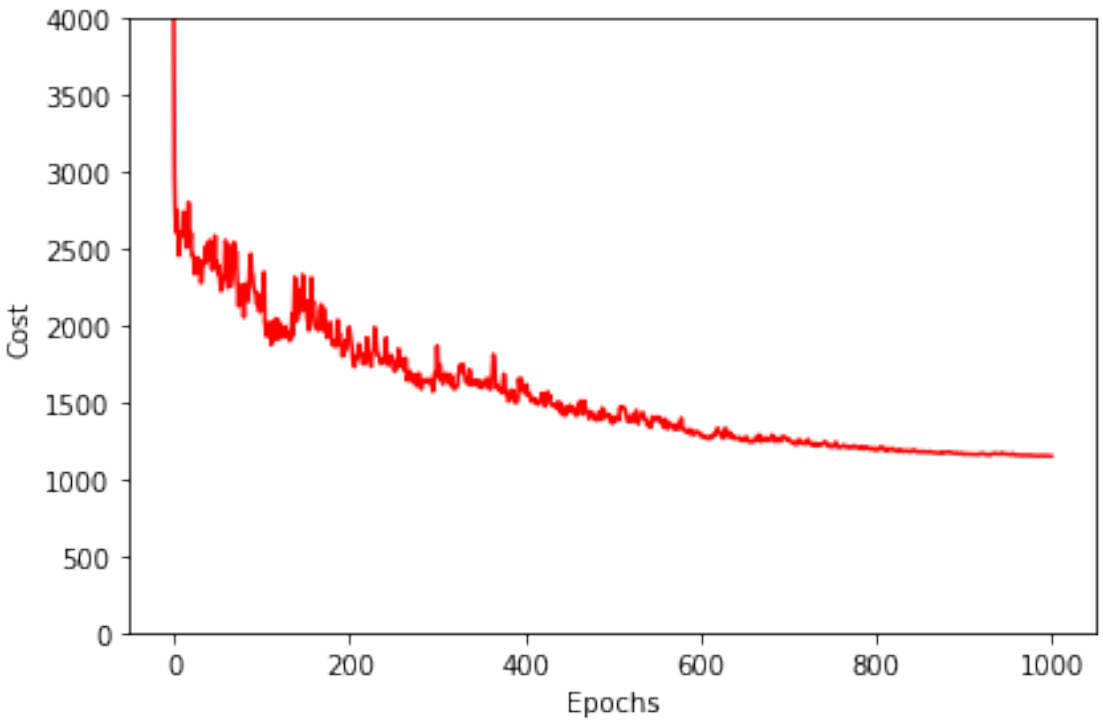
7.3.3 Plot Model History

[99]:
```
# ROUGH PLOT FOR EACH OF THE 50 BATCH RUNS

plt.plot(range(len(nn.cost_)), nn.cost_)
plt.ylim([0, 4000])
plt.ylabel('Cost')
plt.xlabel('Epochs * 50')
plt.tight_layout()
# plt.savefig('./figures/cost.png', dpi=300)
plt.show()
```



[92]:
```
batches = np.array_split(range(len(nn.cost_)), 1000)
cost_ary = np.array(nn.cost_)
cost_avgs = [np.mean(cost_ary[i]) for i in batches]
```

[100]:
```
plt.plot(range(len(cost_avgs)), cost_avgs, color='red')
plt.ylim([0, 4000])
plt.ylabel('Cost')
plt.xlabel('Epochs')
plt.tight_layout()
#plt.savefig('./figures/cost2.png', dpi=300)
plt.show()
```



Through the history of the models cost function, we can easily see that the model did converge and therefore does not need any more epochs.

7.3.4 Understand Accuracy

Training

[338]:
```
import sys

y_train_pred = nn.predict(X_train)

if sys.version_info < (3, 0):
    acc = (np.sum(y_train == y_train_pred, axis=0)).astype('float') / X_train.
 ↪shape[0]
else:
    acc = np.sum(y_train == y_train_pred, axis=0) / X_train.shape[0]

print('Training accuracy: %.2f%%' % (acc * 100))
```

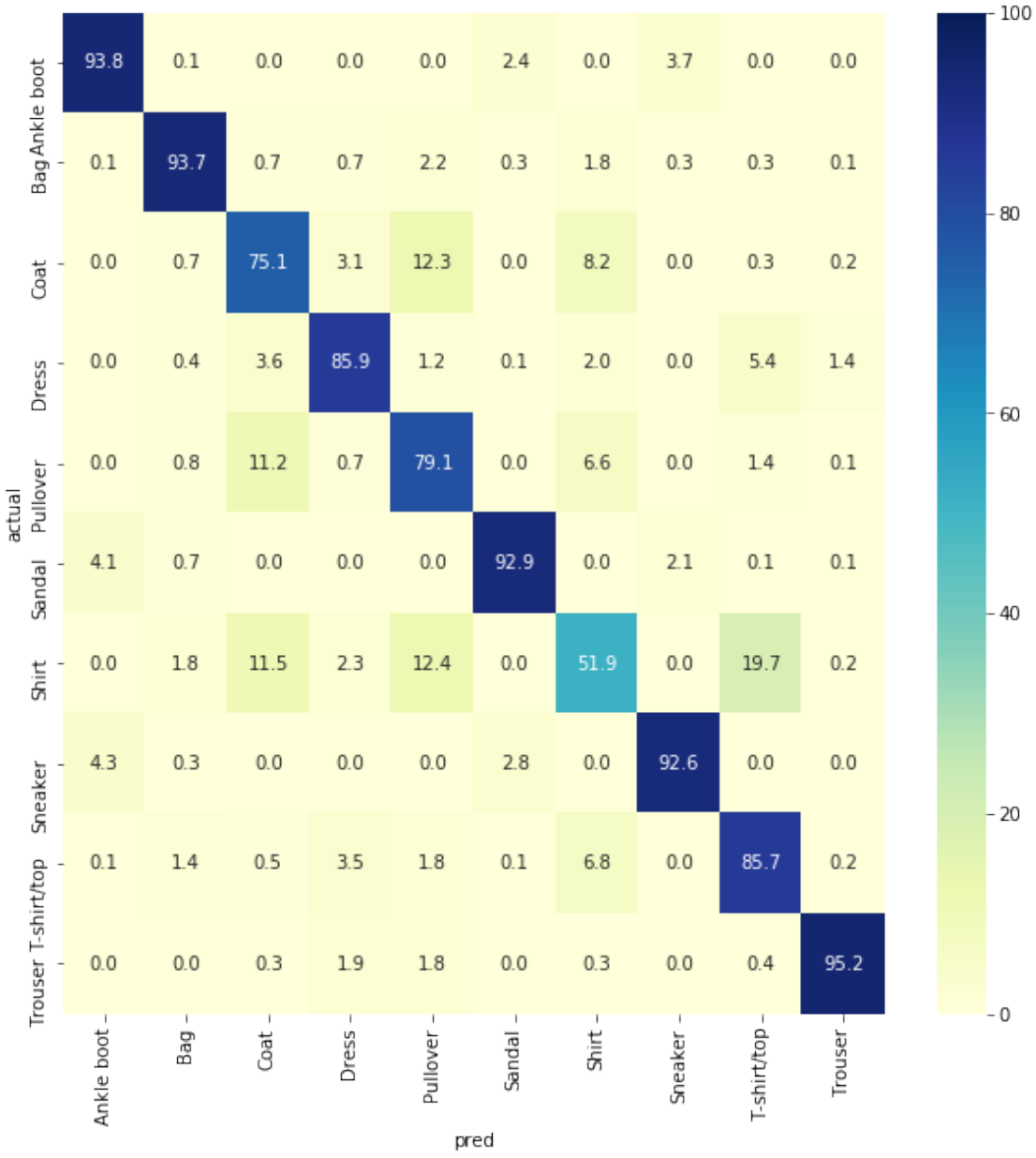
```
Training accuracy: 84.60%
```

[347]:
```
miscl_img = X_test[y_test != y_test_pred][:25]
correct_lab = y_test[y_test != y_test_pred][:25]
miscl_lab= y_test_pred[y_test != y_test_pred][:25]
fig, ax = plt.subplots(nrows=5, ncols=5, sharex=True, sharey=True,)
```

```
fig.set_figheight(10)
fig.set_figwidth(10)
ax = ax.flatten()
for i in range(25):
    img = miscl_img[i].reshape(28, 28)
    ax[i].imshow(img, cmap='Greys', interpolation='nearest')
    ax[i].set_title('%d) t: %s p: %s' % (i+1, labelD[int(correct_lab[i])],␣
 ↪labelD[int(miscl_lab[i])]))

ax[0].set_xticks([])
ax[0].set_yticks([])
plt.tight_layout()
# plt.savefig('./figures/mnist_miscl.png', dpi=300)
plt.show()
```

[311]:
```
acc_df_train_nn, acc_counts_train_nn, acc_perc_train_nn = predAccuracy(y_train,␣
↪y_train_pred)
```

[312]:
```
plt.figure(figsize=(10,10))
ax = sns.heatmap(acc_perc_train_nn, vmin=0, vmax=100, annot=True, fmt=".1f",␣
↪cmap="YlGnBu")
```



In the training set of the neural network model, we see an much better prediction accuracy of 84.6%. The earlier problem of items resembling a pullover minimally being predicted as a pullover seems to be worked out in the neural network. Here the biggest error comes from shirts being

predicted as T-shirts/Tops which is the problem that we outlined from the beginning.

Test

[313]:
```
y_test_pred = nn.predict(X_test)

if sys.version_info < (3, 0):
    acc = (np.sum(y_test == y_test_pred, axis=0)).astype('float') / X_test.
 ↪shape[0]
else:
    acc = np.sum(y_test == y_test_pred, axis=0) / X_test.shape[0]

print('Test accuracy: %.2f%%' % (acc * 100))
```

```
Test accuracy: 82.75%
```

[315]:
```
acc_df_test_nn, acc_counts_test_nn, acc_perc_test_nn = predAccuracy(y_test,␣
 ↪y_test_pred)
```

[283]:
```
# plt.figure(figsize=(10,10))
# ax = sns.heatmap(acc_perc_test_nn, vmin=0, vmax=100, annot=True, fmt=".1f",␣
 ↪cmap="YlGnBu")
```

The test accuracy of the neural network also went up by more than 10%. It does look like we are starting to match the complexity of the dataset with out model.

7.4 Build Keras Convolutional Neural Network

Keras neural network package is the state-of-the-art package for neural networks. Then, for computer vision, covolutional neural networks are the tool of choice in the package. In this model infrastructure, we created a deep neural network with many convolutional layers.

7.4.1 Build Model

[210]:
```
IMG_SIZE = 28

model = Sequential()

# First Layer
model.add(Conv2D(32, kernel_size = (3, 3), activation='relu',␣
 ↪input_shape=(IMG_SIZE, IMG_SIZE, 1)))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())

# Second Layer
model.add(Conv2D(64, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
```

```
# Third Layer
model.add(Conv2D(128, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())

model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(len(np.unique(y_train)), activation = 'softmax'))
```

7.4.2 Model Structure

[350]:
```
print(model.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_62 (Conv2D)           (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d_49 (MaxPooling (None, 13, 13, 32)        0
_________________________________________________________________
batch_normalization_46 (Batc (None, 13, 13, 32)        128
_________________________________________________________________
conv2d_63 (Conv2D)           (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_50 (MaxPooling (None, 5, 5, 64)          0
_________________________________________________________________
batch_normalization_47 (Batc (None, 5, 5, 64)          256
_________________________________________________________________
conv2d_64 (Conv2D)           (None, 3, 3, 128)         73856
_________________________________________________________________
max_pooling2d_51 (MaxPooling (None, 1, 1, 128)         0
_________________________________________________________________
batch_normalization_48 (Batc (None, 1, 1, 128)         512
_________________________________________________________________
dropout_5 (Dropout)          (None, 1, 1, 128)         0
_________________________________________________________________
flatten_5 (Flatten)          (None, 128)               0
_________________________________________________________________
dense_9 (Dense)              (None, 128)               16512
_________________________________________________________________
dense_10 (Dense)             (None, 10)                1290
=================================================================
Total params: 111,370
Trainable params: 110,922
Non-trainable params: 448
_________________________________________________________________
```

```
None
```

7.4.3 Compile and Fit

[211]:
```
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

[212]:
```
y_binary_train = to_categorical(y_train)
y_binary_test = to_categorical(y_test)
```

[351]:
```
history = model.fit(X_train.reshape(len(y_train),28,28,1), y_binary_train,
                    batch_size=50,
                    epochs=50,
                    verbose=1,
                   validation_data=(X_test.reshape(len(y_test),28,28,1),␣
 ↪y_binary_test))
```

7.4.4 Save Model

[272]:
```
model.save(path + 'keras_model')
```

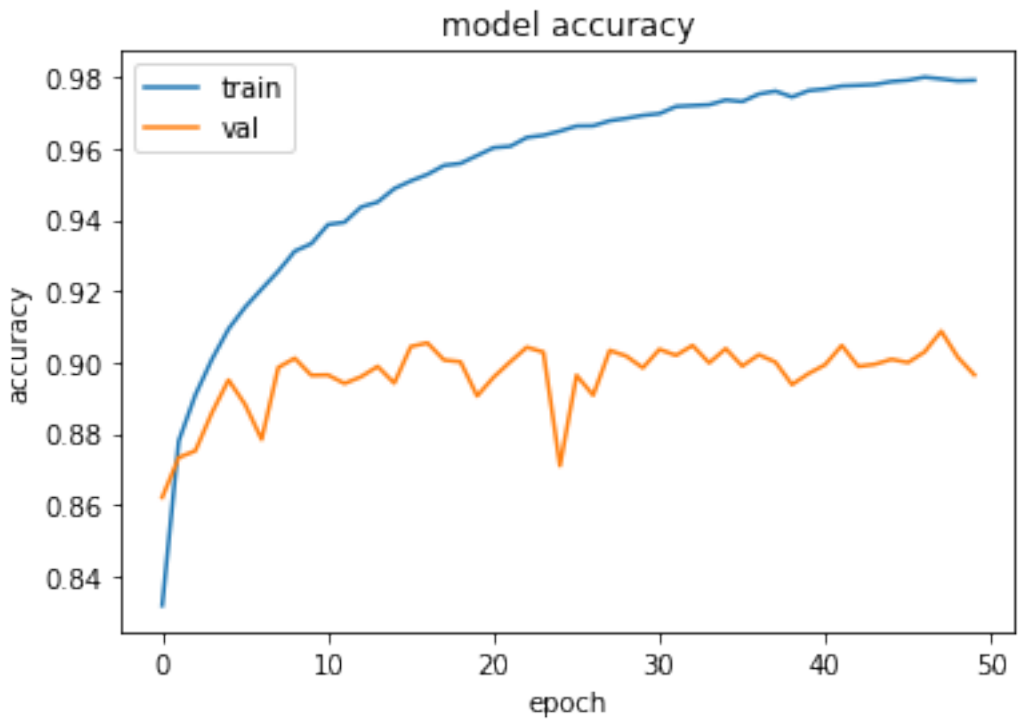
[273]:
```
# model = keras.models.load_model(path + 'keras_model')
```

7.4.5 Look at model History

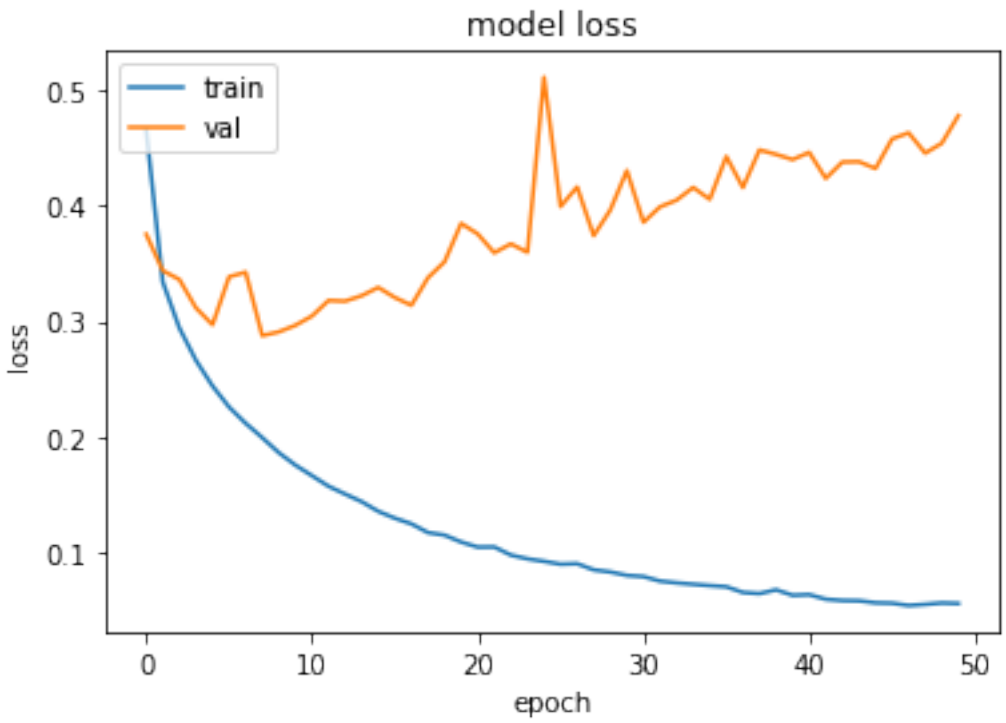
[214]:
```
print(history.history.keys())
```

```
dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```

[215]:
```
# summarize history for accuracy
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```



[216]:
```
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```



The models's accuracy seems to converge after 50 epochs. The validation score is also not affected by the number of epochs, so it looks like we avoided model overfitting.

7.4.6 Understand Model Accuracy

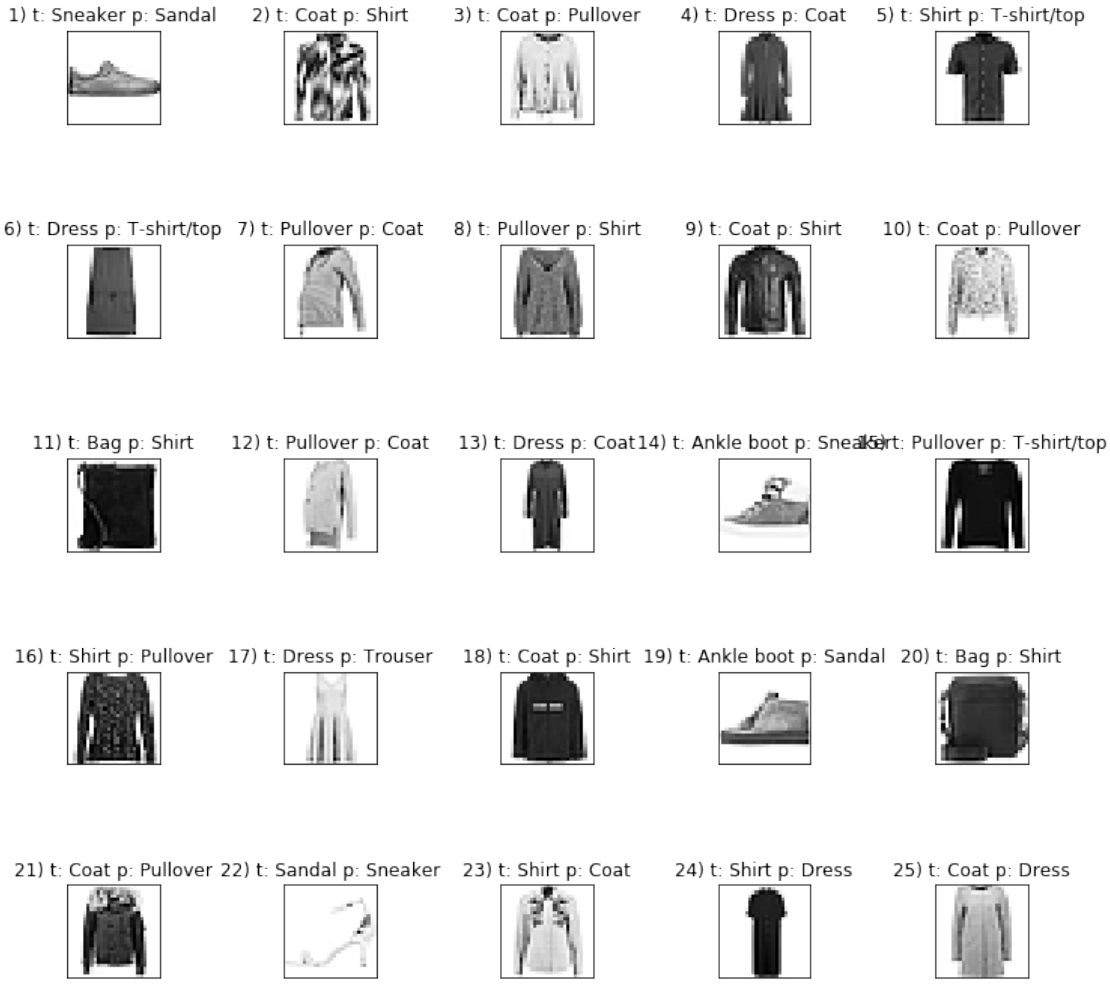
Train

[348]:
```
y_train_pred =np.argmax(model.predict(X_train.reshape(len(y_train),28,28,1)),␣
↪axis=-1)

if sys.version_info < (3, 0):
    acc = (np.sum(y_train == y_train_pred, axis=0)).astype('float') / X_test.
↪shape[0]
else:
    acc = np.sum(y_train == y_train_pred, axis=0) / X_train.shape[0]

print('Test accuracy: %.2f%%' % (acc * 100))
```

```
Test accuracy: 98.62%
```

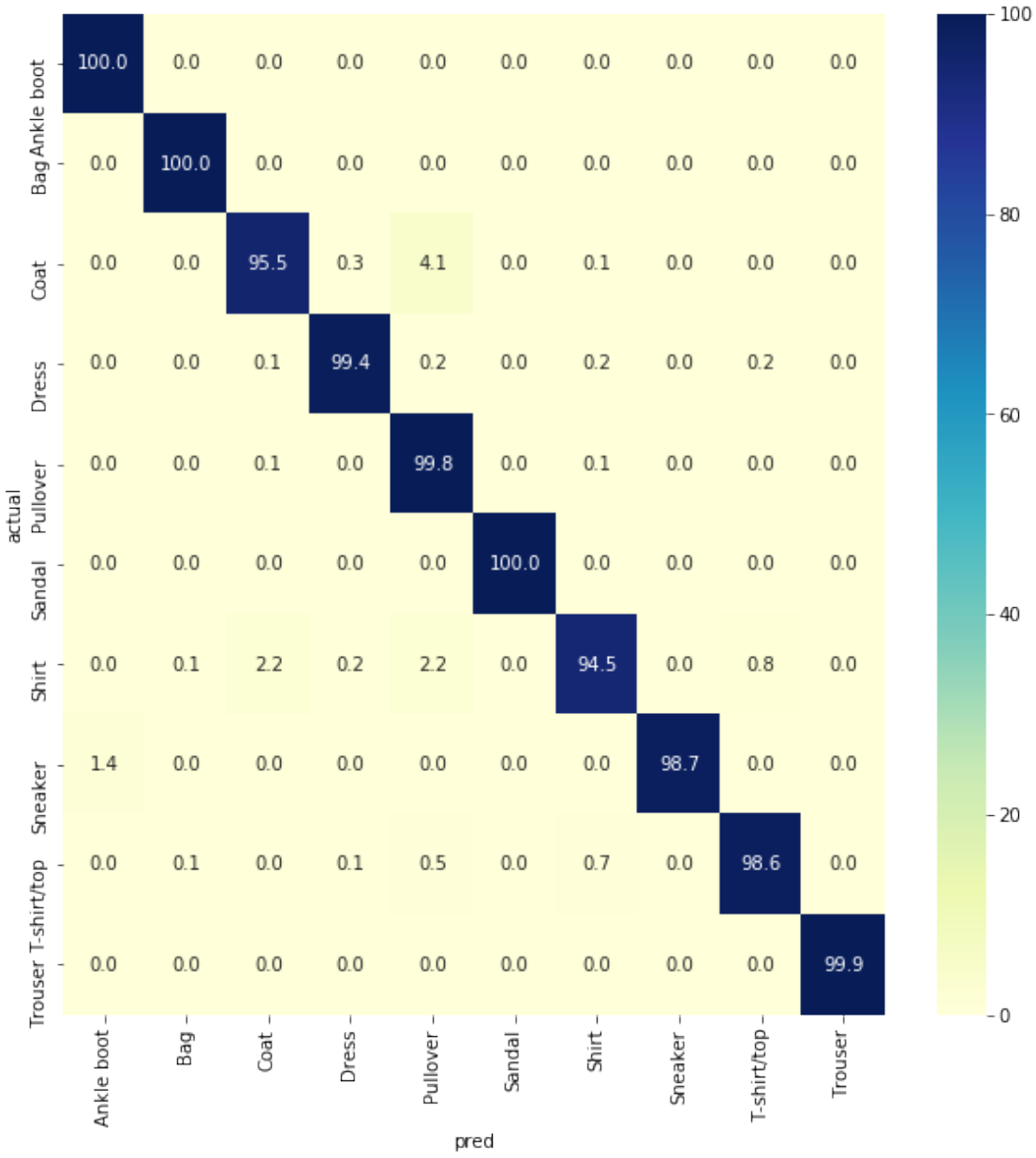
[349]:
```
miscl_img = X_test[y_test != y_test_pred][:25]
correct_lab = y_test[y_test != y_test_pred][:25]
miscl_lab= y_test_pred[y_test != y_test_pred][:25]
fig, ax = plt.subplots(nrows=5, ncols=5, sharex=True, sharey=True,)
fig.set_figheight(10)
```

```
fig.set_figwidth(10)
ax = ax.flatten()
for i in range(25):
    img = miscl_img[i].reshape(28, 28)
    ax[i].imshow(img, cmap='Greys', interpolation='nearest')
    ax[i].set_title('%d) t: %s p: %s' % (i+1, labelD[int(correct_lab[i])],␣
 ↪labelD[int(miscl_lab[i])]))

ax[0].set_xticks([])
ax[0].set_yticks([])
plt.tight_layout()
# plt.savefig('./figures/mnist_miscl.png', dpi=300)
plt.show()
```



[302]:
```
acc_df_train_keras, acc_counts_train_keras, acc_perc_train_keras =␣
 ↪predAccuracy(y_train, y_train_pred)
```

[303]:
```
plt.figure(figsize=(10,10))
ax = sns.heatmap(acc_perc_train_keras, vmin=0, vmax=100, annot=True, fmt=".1f",␣
↪cmap="YlGnBu")
```



In the training set, we see an almost perfect prediction. This could reflect overfitting, but since the validation set still gave us a very good preciction, it looks like the model did match the complexity of the data. In the keras neural network, the biggest error came from predicting Coats as pullovers which is an error that we have seen before.

Test

[304]:
```
y_test_pred =np.argmax(model.predict(X_test.reshape(len(y_test),28,28,1)),␣
↪axis=-1)

if sys.version_info < (3, 0):
    acc = (np.sum(y_test == y_test_pred, axis=0)).astype('float') / X_test.
↪shape[0]
else:
    acc = np.sum(y_test == y_test_pred, axis=0) / X_test.shape[0]

print('Test accuracy: %.2f%%' % (acc * 100))
```

```
Test accuracy: 89.64%
```

[305]:
```
acc_df_test_keras, acc_counts_test_keras, acc_perc_test_keras =␣
↪predAccuracy(y_test, y_test_pred)
```

[306]:
```
# plt.figure(figsize=(10,10))
# ax = sns.heatmap(acc_perc_test_keras, vmin=0, vmax=100, annot=True, fmt=".
↪1f", cmap="YlGnBu")
```

The test accuracy is much higher than the one that we started with. The accuracy went from 71.7% to 89.6% which clearly means our models seem to be getting closer to the optimal answer.

8 Put all Models Together

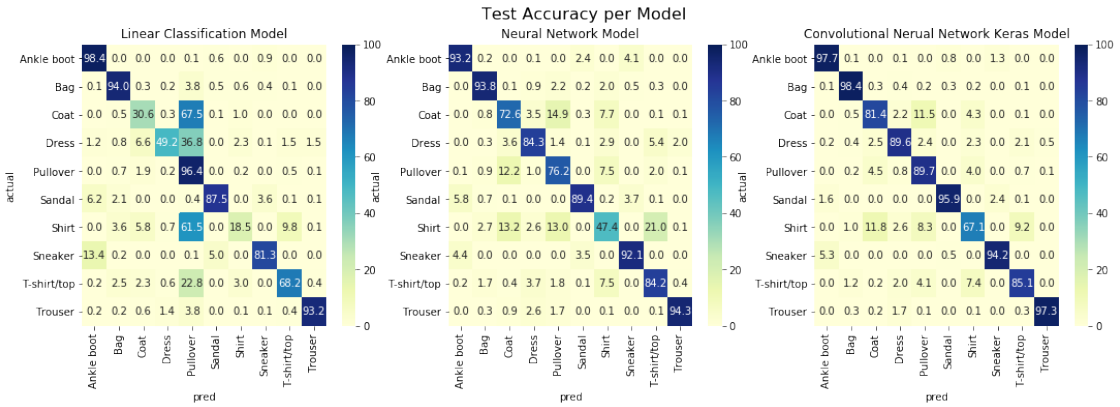
[316]:
```
print('Linear Classification Model Accuracy: ', 100*len(acc_df_test_clf.
↪loc[acc_df_test_clf['Correct']])/len(acc_df_test_clf), '%')
print('Neural Network Model Accuracy: ', 100*len(acc_df_test_nn.
↪loc[acc_df_test_nn['Correct']])/len(acc_df_test_nn), '%')
print('Convolutional Nerual Network Keras Model Accuracy: ',␣
↪100*len(acc_df_test_keras.loc[acc_df_test_keras['Correct']])/
↪len(acc_df_test_keras), '%')
```

```
Linear Classification Model Accuracy:  71.73 %
Neural Network Model Accuracy:  82.75 %
Convolutional Nerual Network Keras Model Accuracy:  89.64 %
```

[321]:
```
fig = plt.figure(figsize=(18,5))
fig.suptitle('Test Accuracy per Model', fontsize=16)

plt.subplot(1,3,1)
ax = sns.heatmap(acc_perc_test_clf, vmin=0, vmax=100, annot=True, fmt=".1f",␣
↪cmap="YlGnBu")
plt.title('Linear Classification Model')
plt.subplot(1,3,2)
```

```
ax = sns.heatmap(acc_perc_test_nn, vmin=0, vmax=100, annot=True, fmt=".1f",␣
 ↪cmap="YlGnBu")
plt.title('Neural Network Model')
plt.subplot(1,3,3)
ax = sns.heatmap(acc_perc_test_keras, vmin=0, vmax=100, annot=True, fmt=".1f",␣
 ↪cmap="YlGnBu")
plt.title('Convolutional Nerual Network Keras Model')
```

[321]:
```
Text(0.5, 1, 'Convolutional Nerual Network Keras Model')
```



[353]:
```
from IPython.display import HTML, display
import tabulate
table = [["Model",'Train Accuracy','Test Accuracy', 'Speed'],
         ["Linear Classification",'73.89%','71.73%', '<1min'],
         ["Neural Network",'84.60%','82.75%', '<10min'],
         ["Keras Convolutional Neural Network",'98.62%', '89.64%', '<1hr']]
display(HTML(tabulate.tabulate(table, tablefmt='html')))
```

```
<IPython.core.display.HTML object>
```

9 Conclusion

All in all, we see that the best model is the Keras Convolutional Neural Network. It gave us a extremly good test accuracy which is much higher than the first model we created. The only downside of the apprach is the training speed. In that regard, the linear classification model has the upperhand as it is more than 100 times faster to train.

[ ]: