Capstone Project #2

Fashion MNIST Classification with Neural Network Models

Shiva Safaei –Final Report

• Problem Statement

Have you ever seen something which made you wonder whether you can trust your eyes? Perhaps a blue dress which your friend says it is gold –that 2015 controversy befuddled the world. Or completely parallel lines which appear to be bent. Or even image of completely steady circle which look as if they are moving when you move your head, etc.
Without a double everybody has experienced such moments in their lives. Certainly life would be easier if we could have machines 'see and judge' for us without human eyes optical illusions.Luckily in the past decades image processing techniques have advanced so much that getting computers to perceive visual information (images and videos) is now a trivial reality.
In computer science image processing is the use of computer algorithms to perform processing on digital images. Its applications range from entertainment, medicine, environment, agriculture, military, industry, passing by geological processing and remote sensing. Image processing (along with natural language processing, speech recognition, etc.) is a subclass of deep learning (also known as deep structured learning or hierarchical learning) which is part of a broader family of machine learning methods based on artificial neural networks. Deep learning uses multiple layers to progressively extract higher level features from the raw input. In image processing, lower layers may identify edges, while the higher layers may identify the concepts relevant to a human such as digits or letters or faces.
Using neural networks, specifically, Convolutional Neural Networks (CNN)s, we want to answer these questions: *how accurate can a computer automatically detect pictures of fashion items? And how can we increase the accuracy?* Assuredly the *whether* question has been answered long ago for as a simple case as detecting a cat versus a dog to more sophisticated tumor cell recognition problems.

Although fashion-MNIST image classification has its own demanding applications, it is presented here to showcase the methodology and how to enhance it. These methods could be applied in a wide scope of classification problems.

• Description of Data

The MNIST database (Modified National Institute of Standards and Technology database) is a database of handwritten digits that is commonly used for training various image processing systems.
Recently, Zalando Research has released a new class of MNIST, called Fashion-MNIST dataset which intended to serve as a direct replacement for the original MNIST to benchmark machine learning algorithms. It is consisting of a training set of $60,000$ examples and a test set of $10,000$ examples. Each example is a 28×28 grayscale image, associated with a label from 10 classes (Table **??**). Each training and test case is associated with one of ten labels (0-9). The data can be downloaded from Kaggle or GitHub.

Table 1: Table of Fashion-MNIST; the definition of classes

Class Number	Item
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle Boot

∗ Dataframes:

We can sart with defining the `df_train` and `df_test` dataframes representing the train and test datasets whoes ecah 60,000 and 10,000 rows indicates one image. Each image has 784 pixels (28 pixels in height and 28 pixels in width), and has a single pixel-value associated with it, indicating the lightness or darkness of that pixel. The pixel-values are integer numbers between 0 and 255 stored in columns 2 to 785. The first column corresponds to the class label of images.

∗ Data preparation:

Our goal is to build an accurate model using the train dataset and test it by applying it

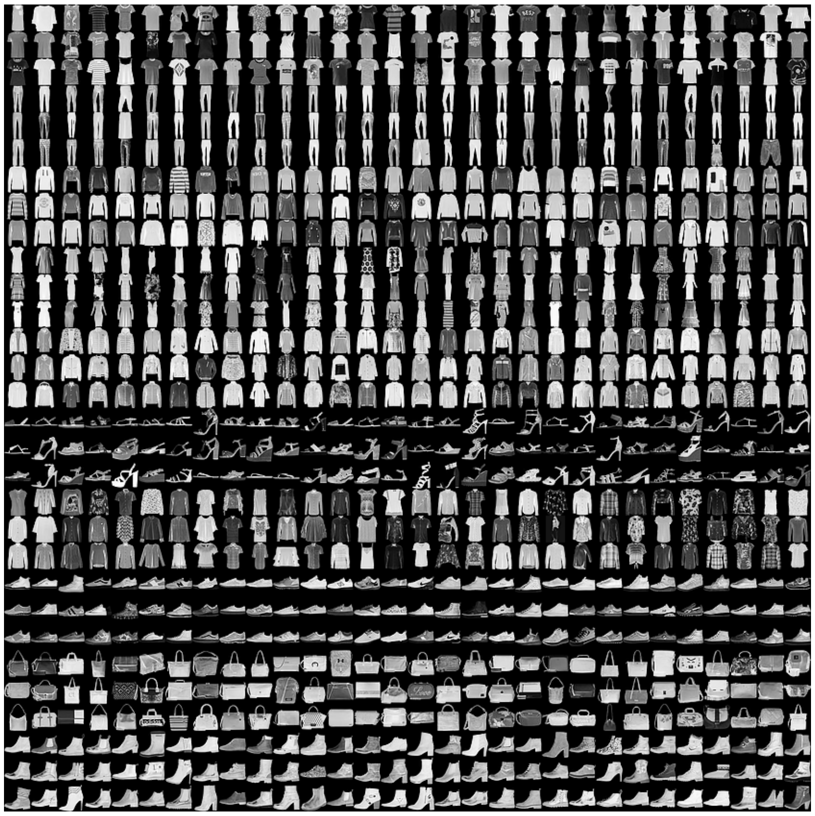


Figure 1: fashion-mnist-sprite

on the test dataset which the model has never seen before. For preparing the data we can take these extra steps:

- One common practice to eliminate overfitting is to split the original training dataset to 80% train and 20% validation chunks.
- We can rescale images by dividing pixel-values by 255, so the features range between 0 to 1.
- Fortunetly the data is clean without particular outliers or missing values, thus no data wrangling/cleaning is performed.

• Visualization

∗ Image visualization:

Our dataset contains pixel-value columns, hence the first natural way to visualize the data is to display the images. Fig. **??** show randomly selected sample images representing each class.

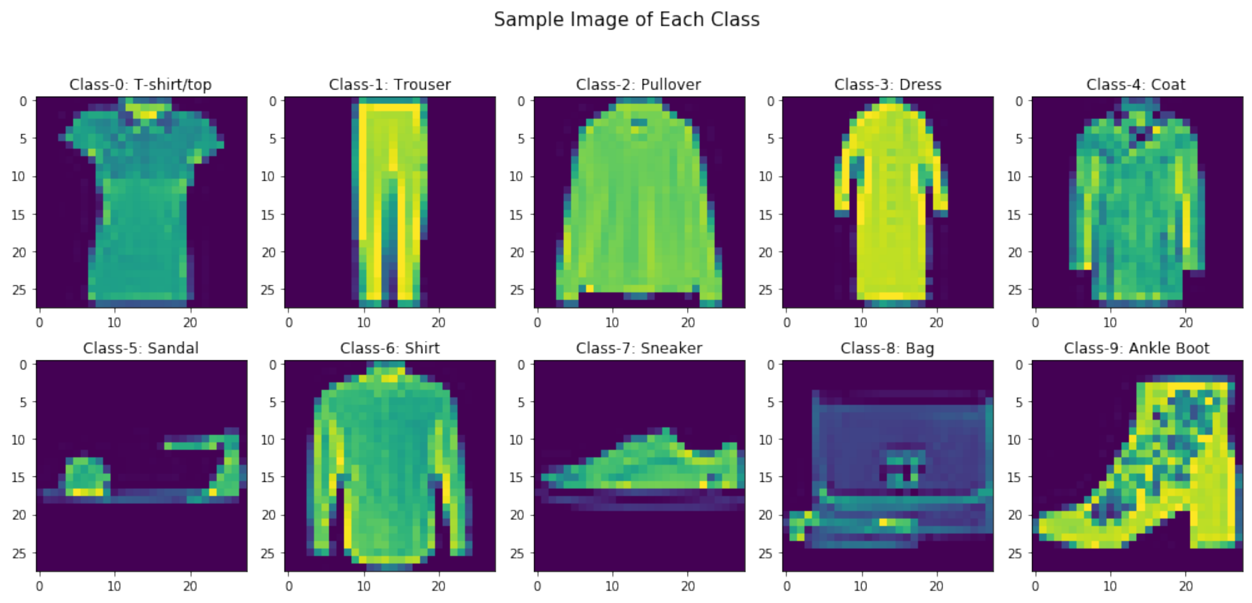
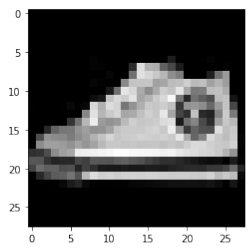
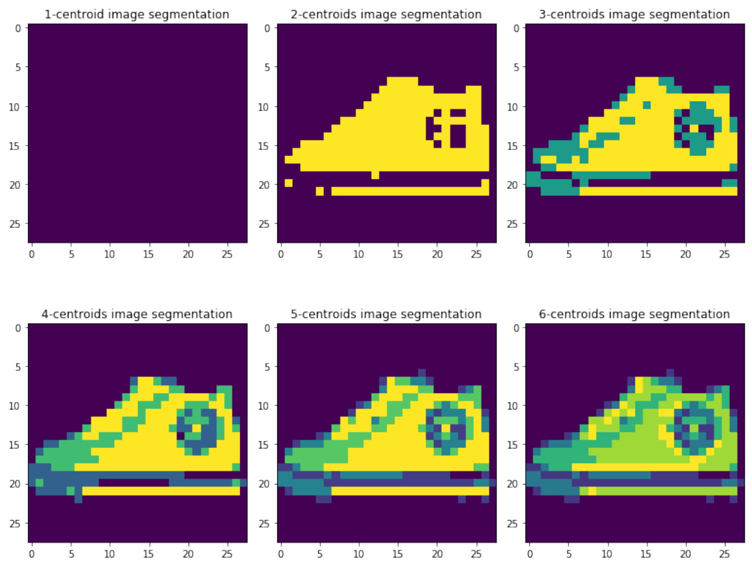


Figure 2:

∗ Image segmentation:

Now that we visualized the images and know how each class looks like, we can further analyze them. Every digital image is composed of pixels, and pixels form boundaries (curves, lines, etc.) with specific patterns. We can classify the pixels based on their certain characteristics in order to obtain simplified segmentations of an image. The process of partitioning a digital image into multiple segments (sets of pixels, also known as image objects) is called image segmentation.
The result of image segmentation is a set of segments that collectively cover the entire image, or a set of contours extracted from the image. Each of the pixels in a region are similar with respect to some characteristic or computed property, such as color, intensity, or texture. Adjacent regions are significantly different with respect to the same characteristic(s). Image segmentation has a vast area of applications from face detections to medical imaging for locating tumors, etc.
Here I use an unsupervised learning technique (`KMean` algorithm) to show the first 6 segmentations of a random image from train dataset (Fig. **??** and Fig. **??**).
The original image has been categorized into `K`-centroid segmented images. The first one (1-centroid) has only one cluster, which means all pixels are classified into one group, thus we have one blank image. By increasing the number of `K` (centroids) different groups of pixels emerged, so we can see more detailed edges and borders of the image.


(a) A sample image



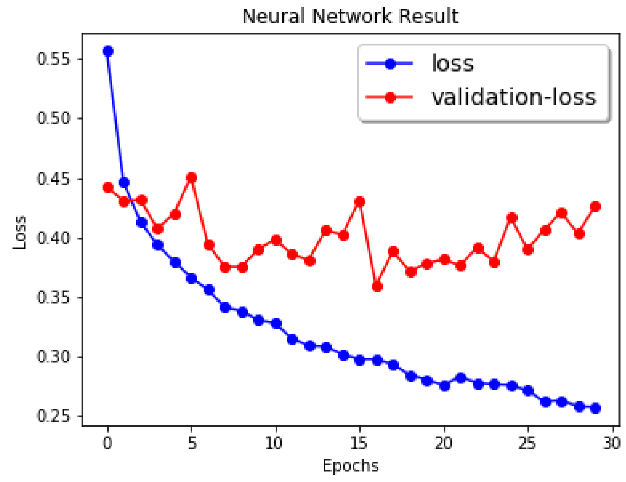
(b) The first 6 segmentations of the sample image

Figure 3: Image segmentations

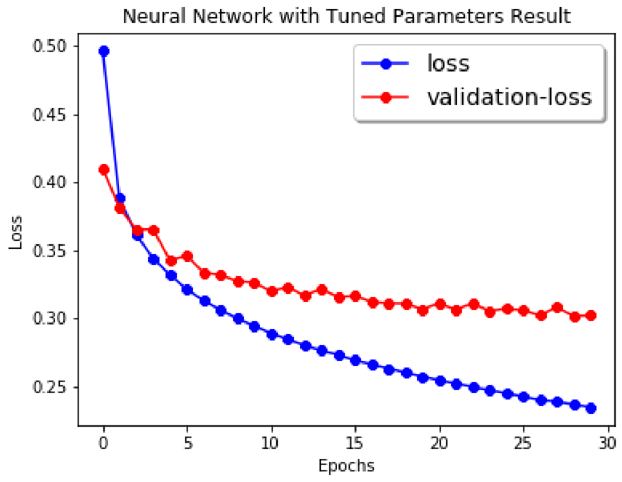
• Multilayer Neural Network

The objective of the fashion-MNIST project is to develop a model that identifies the fashion items –it can be translated into an image classification problem for deep learning models. Due to its wide range of applications, such a classification became quite popular. Subsequently, there has been already numerous studies in this field that build models with accuracy somewhere between 80 to 90+ percent.
My goal is to find a better model by tweaking the parameters combined with relevant statistical algorithms, to enhance the accuracy without a high computational cost.
I started with image classification using a basic neural network. Our model consist of 3 sequential layers; the first layer has 100 nodes and takes the input features and adds it to the network. In a neural network, we would update the weights and biases of the neurons on the basis of the error at the output. Activation functions make this process (back-propagation) possible as it decides, whether a neuron should be activated or not by calculating a weighted sum and further adding bias with it. The activation function does the non-linear transformation to the input making it capable to learn and perform more complex tasks. A neural network without an activation function is essentially just a linear regression model.
For the first two layers `ReLU` activation function is used –it is a linear function for all positive values, and zero for all negative values. The last layer has 10 nodes corresponding to the 10 classes in our dataset and uses the Softmax activation function that returns their probabilities.
Once the model is built we can configure its learning process with `compile()`. Since

our target (y) is converted to categorical values we use `categorical_crossentropy` loss function. An optimizer is the second required argument for compilation; here I used stochastic gradient descent (`SGD`) optimizer with a learning rate of 0.05 and momentum of 0.9 (to accelerate `SGD` in the relevant direction and dampens oscillations).
Training is performed for 30 epochs and the test set is evaluated at the end of each epoch. The learning curve plot is shown in the Fig. **??**. The accuracy of the neural network model is **85.99%**.



(a)



(b)

Figure 4: Plots of loss function for the Neural Network models

* Hyperparameter tuning:

The neural network has many parameters and tuning these parameters and using the right ones could enhance the performance of a model. However, since the number of parameters is very large, it is quite difficult to configure all of them at once. In scikit-learn the hyperparameter optimization is provided in `GridSearchCV` class. Here we will employ the same technique to grid search and find *some* of the best parameters for our neural network model. It is worth mentioning adding more parameters to the grid search, exponentially increases the calculation time. Thus, I only grid searched for the optimizer and activation functions. Keras models can be used in scikit-learn by wrapping them with the `KerasClassifier` or `KerasRegressor` class.
The result of our grid search is:
`best params: {'activation': 'tanh', 'optimizer': 'Adagrad'}`.
Fig. **??** shows the new result using these parameters. The accuracy of the hyper tuned neural network is **89%**, which is about 3% better that what is was previously (85.99%). Not quite impressive considering the time we spent to find the two parameters. In the following, we explore the convolutional nueurl network model.

• Convolutional Neural Network

A convolutional neural network (CNN) is a neural network that has at least one conventional layer. Analogous to the neural network which is inspired by a human brain, in CNN the connectivity pattern between neurons resembles the organization of the human/animal visual cortex. The convolutional layer applies the convolution with a learnable filter (`kernel`), as a result the network learns the patterns in the images: edges, corners, arcs, then more complex figures. Such layer receives its input from multiple units from the previous layer which together create a proximity. Therefore, the input units (that form a small neighborhood) share their weights.
Each convolutional layer within a neural network should have the following attributes:

- Input is a tensor with shape (number of images)×(image width)×(image height)×(image depth).
- Convolutional kernels whose width and height are hyperparameters, and whose depth must be equal to that of the image. Convolutional layers convolve the input and pass its result to the next layer. This is similar to the response of a neuron in the visual cortex to a specific stimulus.

The following explains the layer architecture of our cnn model:

1- The first important step is to reshape the input, which is a (number-of-images)×28×28×1 array of pixel values. Image depth is 1 because they are gray scale.

2- The first layer in a CNN is always a convolutional layer. The kernel size refers to the width×height of the filter mask. `padding='same'` adds the required padding to the input image to ensure that the output has the same shape as the input, and `strides=1` means the kernel is moved by 1 as it passes over the image. These two eliminate losing information on corners of images.

3- `MaxPooling2D` reduces the dimensions of the image by combining the outputs of neuron clusters at one layer into a single neuron in the next layer (down-sampling).

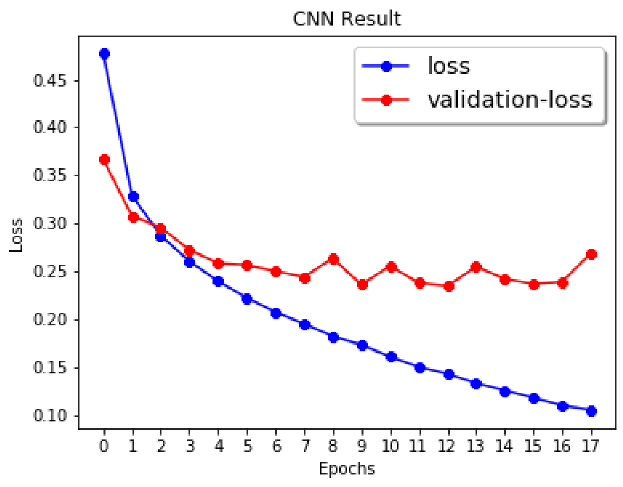
4- A fully connected layer occupies most of the parameters, thus it is prone to overfitting. One method to reduce overfitting is `Dropout`, in which individual nodes are either dropped out of the net with probability `1-p` or kept with probability `p`, so that a reduced network is left.

5- Finally, before passing our pooled feature map to the neural network (the next step), we need to convert it into a column like data by using `Flatten()`.

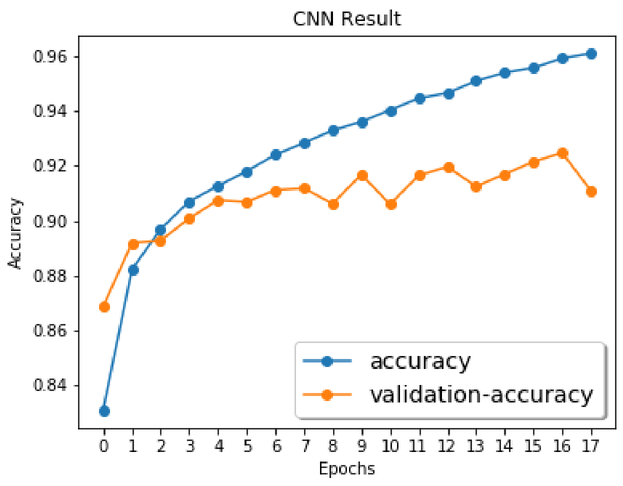
6- The last step is a layer of 10 nodes for classifying our 10 calss output.

7- Now, we can compile the model.

8- A `callback` is added to the fit step in order to monitor validation accuracy to eliminate overtraining (it stops training when the monitored quantity has stopped improving).



(a)



(b)

Figure 5: Plots of loss function (a) and accuracy (b) of the CNN model

The loss and accuracy of the cnn model are shown in Fig. **??** and Fig. **??**. The accuracy of the cnn model is **91.81%**.

Clearly, the convolutional neural network does a better predicting job as compared to the neural network. The reason for that is rooted in the fundamentals of these two models.
A traditional neural network with fully connected nodes suffers from the *curse of dimensionality* which does not scale well with high resolution images, e.g., a 1000×1000-

pixel image with RGB color channels has 3 million weights, which is too high to feasibly process efficiently at scale with full connectivity.
Also, such network architecture does not take into account the spatial structure of data, treating input pixels which are far apart in the same way as pixels that are close together. This ignores locality of reference in image data, both computationally and semantically. Thus, full connectivity of neurons is wasteful for the purpose of image recognition that is dominated by spatially local input patterns.
On the other hand cnn models as opposed to multilaye neural network exploit the strong spatially local correlation present in natural images. This means that they are able to reduce dramatically the number of operation needed to process an image by using convolution on patches of adjacent pixels, because adjacent pixels together are meaningful.
A cnn model uses convolution of image and filters to generate invariant features which are passed on to the next layer. The features in next layer are convoluted with different filters to generate more invariant and abstract features and the process continues till one gets final feature which is invariant to occlusions. Process of capturing the features is the key to success of cnn models.

∗ Image Augmentation:

Arguably, the data which is used during the training step plays the most important role in the accuracy of a model. In fact, the performance of the neural networks improves with the amount of data available. In our model, in order to to (artificially) expand the size of the training dataset we can create modified versions of images in the dataset; this is a well-known technique, called image augmentation. The purpose of image augmentation is to not only increase the quantity of the dataset, but also to enrich its diversity. This method generates transformed versions of images in the training dataset that belong to the same class as the original image. The transformation includes a range of operations from the field of image manipulation, such as shifts, flips, zooms, etc.
`ImageDataGenerator` class provides the ability to use data augmentation automatically when training a model. Here I used the following arguments for building new training and validation datasets:

· `rotation_range` for rotating images
· `width_shift_range` and `height_shift_range` for shifting images
· `shear_range` for shearing the images (in counter-clockwise direction in degrees)
· Image zoom via the `zoom_range` argument

The train and validation generator is created by `flow()` which can be used to train our model by calling the `fit_generator()` function. `steps_per_epoch` indicates indicates how many times a new batch is fetched from the generator during single epoch. The accuracy of the image-augmented cnn is **92.12%**.

∗ KFold Cross-Validation:

So far we built two models (multilater neural network and cnn) by splitting our data into training and validation sets. The training set is used to train the model, and the validation set is used to validate it on the data that the model has never seen before (the split is 80%-20%).
With such a split the accuracy is high during the training process, however, it usually fails when it tests the test dataset. This is because through training the model tries to be be exact and includes all the noises of the training dataset, then it is validated on a small validation subset, and at the end when it sees a new big test dataset due to its variability the model does not do as well. This is a typical overfitting problem.
One way to overcome this problem is to diversify the dataset as much as possible (as I explained in image augmentation section), another way is to not use the entire dataset when training the learner. As such, some of the data is removed before training begins. Then when training is done, the data that was removed can be used to test the performance of the learned model on the new data. This is the basic idea for a whole class of model evaluation methods called cross validation.
In cross validation the dataset is divided into k equal size subsets. One of the k subsets is retained as the validation data for testing the model, and the remaining k-1 subsubsets are used as training data. This process is then repeated k times, with each of the k subsets used exactly once as the validation data.
Here, I utilized this strategy to yet improve our model; first I create a `molde_eval` function with the exact same parameters as the original cnn (hereafter called cnn_1). Then I make a 10-fold split of the data which will be fed to the `molde_eval` 10 times.

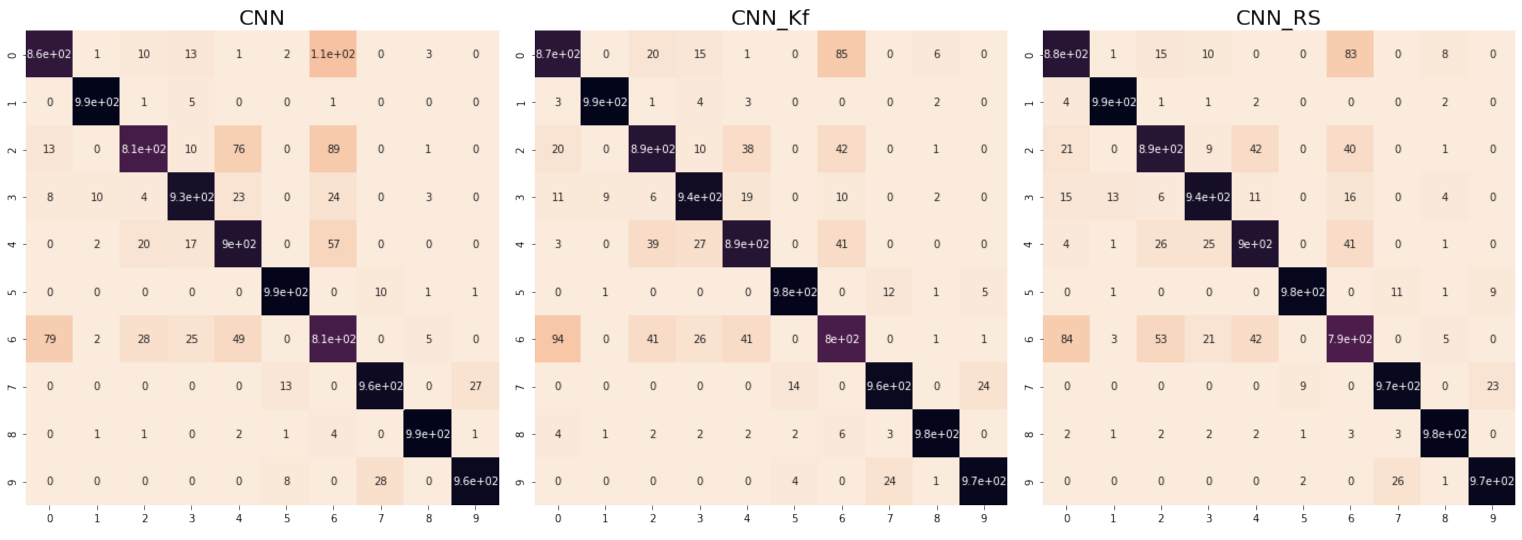


Figure 6: Confusion matrices of the cnn models

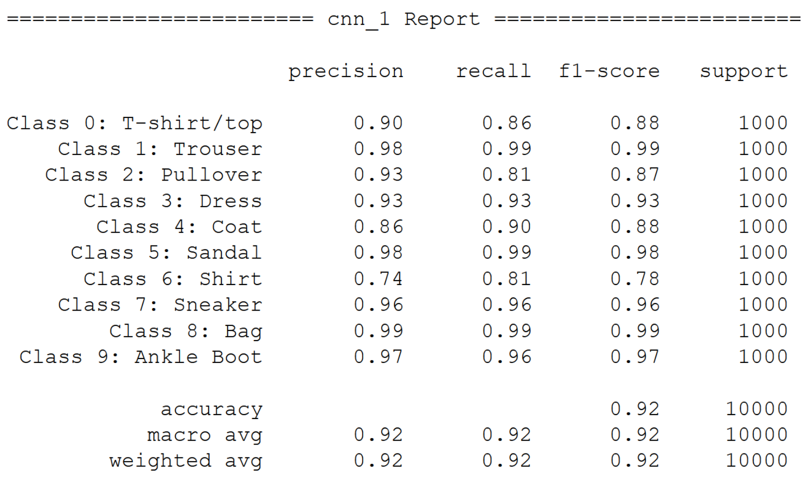
In each iteration the model is generating the best weight and saves that as the initial weight for the next iteration.
The first cnn_kf model splits the train detaset into 10 consecutive folds in which every sample is in one and only one fold. The next cnn_rs is also dividing the train dataset to 10 subsets, but instead of 10 folds containing only once-participating samples here we take random permutation of them. The accuracy of Kfold cnn and shuffle split Kfold cnn are **92.7%** and **92.9%** respectively.

• Classification Report

Eventually, we can compare and summarize the performance of our models by checking their predictions (Fig. **??**). `classification_report` takes the true labels (`y_true`), predicted labels, and the list of target classes and returns the classification metrics;

- `pecision` is the ratio of correctly predicted positive observations to the total predicted positive observations.
- `recall` is the ratio of correctly predicted positive observations to the all observations in actual class.
- `f1-score` is the weighted average of Precision and Recall.
- `support` is the number of samples of the true response that lie in that class.
- `macro avg` and `weighted avg` are unweighted and weighted average of the precision of all the classes (since the number of samples in each class is equal these, two parameters are also equal).

Noticeably the classifiers which are trained by Kfolds cross validation are doing a better job than the cnn_1 model. Although, class 6 is 4% improved in cnn_kf and cnn_rs compared to cnn_1, it still has the lowest precision amongst all other classes.
For better presenting the predictions the confusion matrix of the models are shown in the Fig. **??**. The diagonal elements represent the number of points for which the predicted label is equal to the true label, while off-diagonal elements are those that are mislabeled by the classifier. The higher the diagonal values of the confusion matrix the better, indicating many correct predictions. x and y axis show the predicted and true labels respectively.
It is apparent that the most common mistake is that class 6 (Shirt) was mislabeled to be class 0 (T-shirt/top) and vice versa. It somehow makes sense as shirts and T-shirts/tops are similar!
A couple of mislabeled images in our cnn_1 and cnn_kf models are shown in Fig. **??** and Fig. **??**. By looking at these images it is evident that wrong predictions are inevitable, as some of the images are *SO* similar to many classes at the same time (for instance the third image of the second row above could definitely be a shirt or a dress)!!!



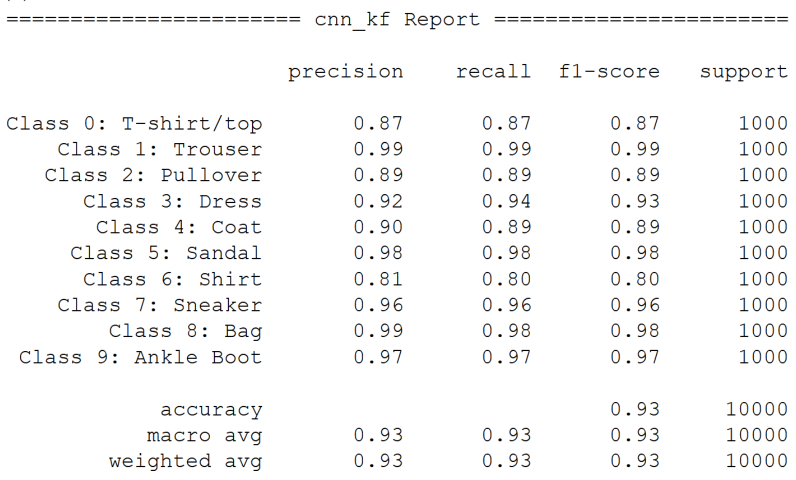
```
======================== cnn_1 Report ========================

                      precision    recall  f1-score   support

Class 0: T-shirt/top       0.90      0.86      0.88      1000
    Class 1: Trouser       0.98      0.99      0.99      1000
   Class 2: Pullover       0.93      0.81      0.87      1000
      Class 3: Dress       0.93      0.93      0.93      1000
       Class 4: Coat       0.86      0.90      0.88      1000
     Class 5: Sandal       0.98      0.99      0.98      1000
      Class 6: Shirt       0.74      0.81      0.78      1000
    Class 7: Sneaker       0.96      0.96      0.96      1000
        Class 8: Bag       0.99      0.99      0.99      1000
 Class 9: Ankle Boot       0.97      0.96      0.97      1000

            accuracy                           0.92     10000
           macro avg       0.92      0.92      0.92     10000
        weighted avg       0.92      0.92      0.92     10000
```

(a)



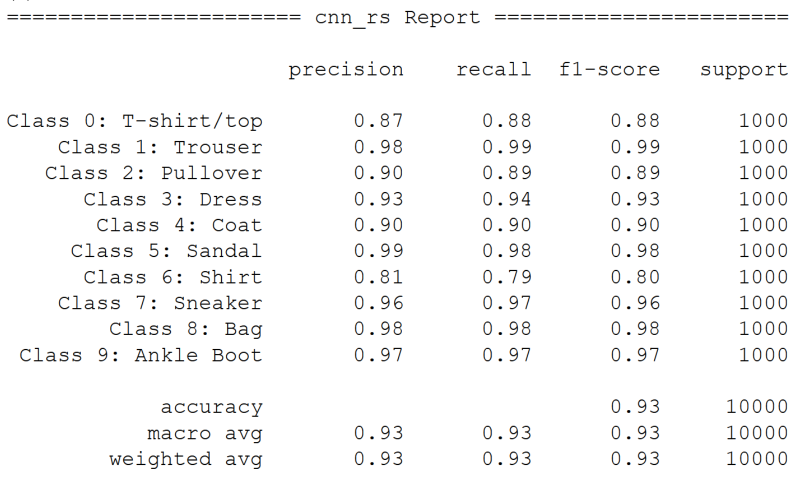
```
======================= cnn_kf Report =======================

                      precision    recall  f1-score   support

Class 0: T-shirt/top       0.87      0.87      0.87      1000
    Class 1: Trouser       0.99      0.99      0.99      1000
   Class 2: Pullover       0.89      0.89      0.89      1000
      Class 3: Dress       0.92      0.94      0.93      1000
       Class 4: Coat       0.90      0.89      0.89      1000
     Class 5: Sandal       0.98      0.98      0.98      1000
      Class 6: Shirt       0.81      0.80      0.80      1000
    Class 7: Sneaker       0.96      0.96      0.96      1000
        Class 8: Bag       0.99      0.98      0.98      1000
 Class 9: Ankle Boot       0.97      0.97      0.97      1000

            accuracy                           0.93     10000
           macro avg       0.93      0.93      0.93     10000
        weighted avg       0.93      0.93      0.93     10000
```

(b)



```
======================= cnn_rs Report =======================

                      precision    recall  f1-score   support

Class 0: T-shirt/top       0.87      0.88      0.88      1000
    Class 1: Trouser       0.98      0.99      0.99      1000
   Class 2: Pullover       0.90      0.89      0.89      1000
      Class 3: Dress       0.93      0.94      0.93      1000
       Class 4: Coat       0.90      0.90      0.90      1000
     Class 5: Sandal       0.99      0.98      0.98      1000
      Class 6: Shirt       0.81      0.79      0.80      1000
    Class 7: Sneaker       0.96      0.97      0.96      1000
        Class 8: Bag       0.98      0.98      0.98      1000
 Class 9: Ankle Boot       0.97      0.97      0.97      1000

            accuracy                           0.93     10000
           macro avg       0.93      0.93      0.93     10000
        weighted avg       0.93      0.93      0.93     10000
```

(c)

Figure 7: Classification reports of the cnn_1 (a), Kfold cnn (b) and shuffle split Kfold cnn(c)

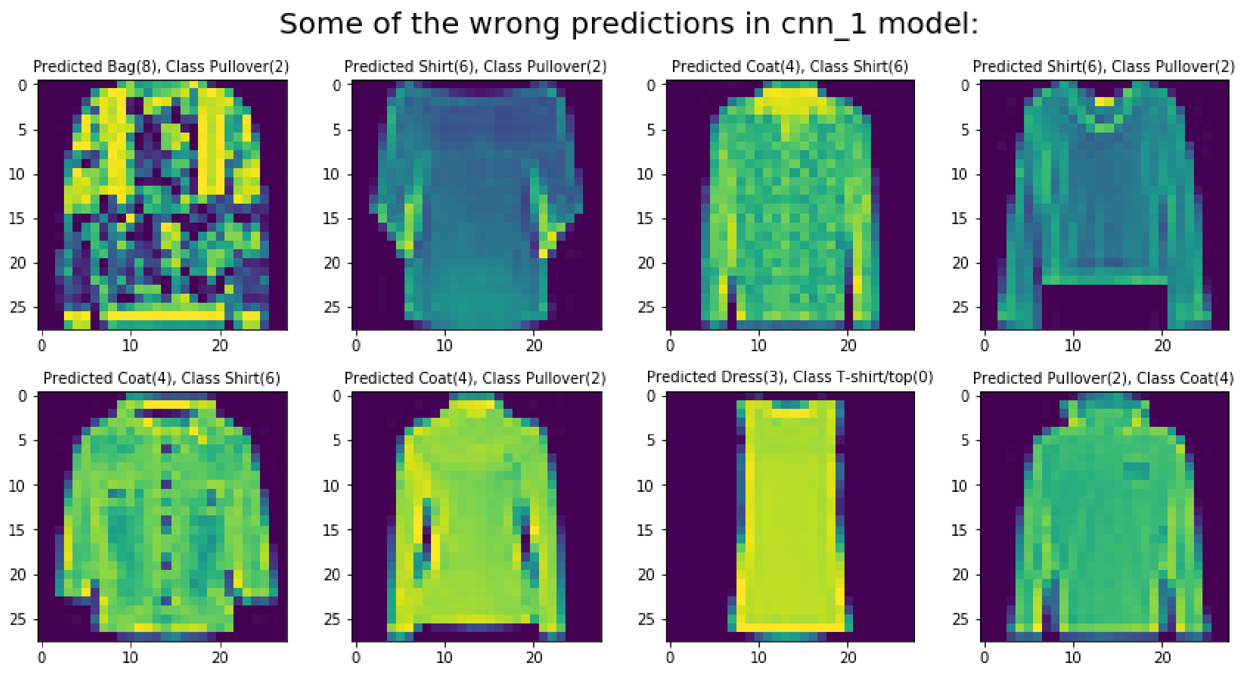

Some of the wrong predictions in cnn_1 model:
Predicted Bag(8), Class Pullover(2)
Predicted Shirt(6), Class Pullover(2)
Predicted Coat(4), Class Shirt(6)
Predicted Shirt(6), Class Pullover(2)
Predicted Coat(4), Class Shirt(6)
Predicted Coat(4), Class Pullover(2)
Predicted Dress(3), Class T-shirt/top(0)
Predicted Pullover(2), Class Coat(4)

Figure 8:

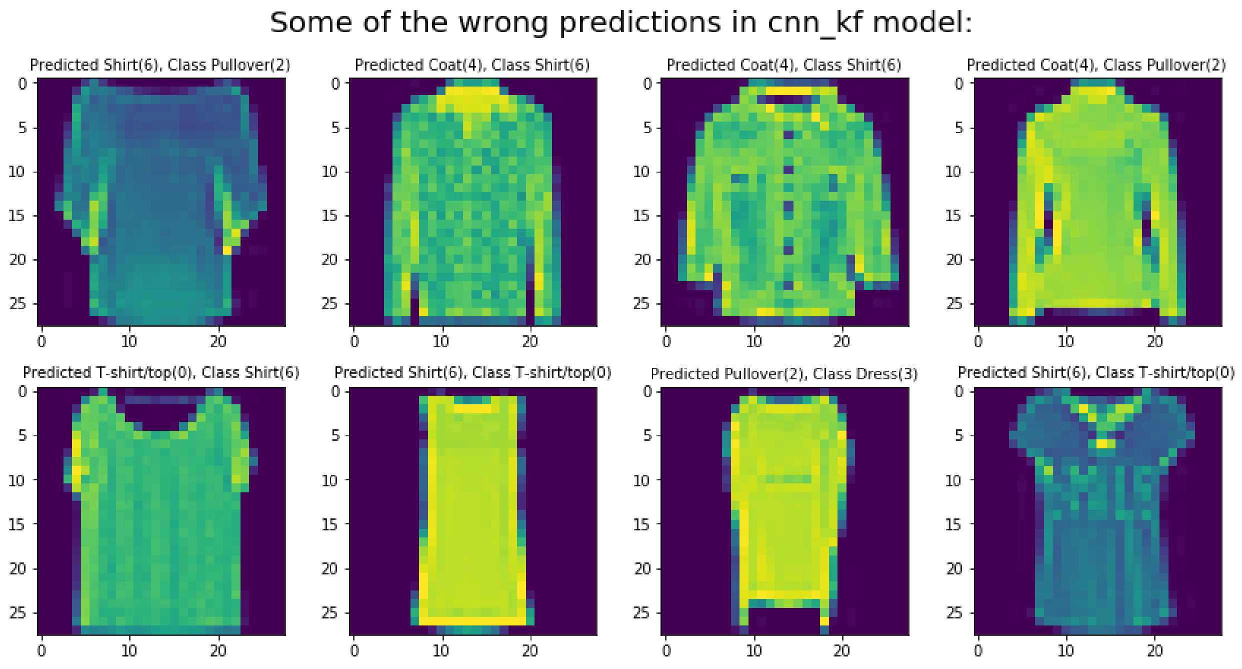
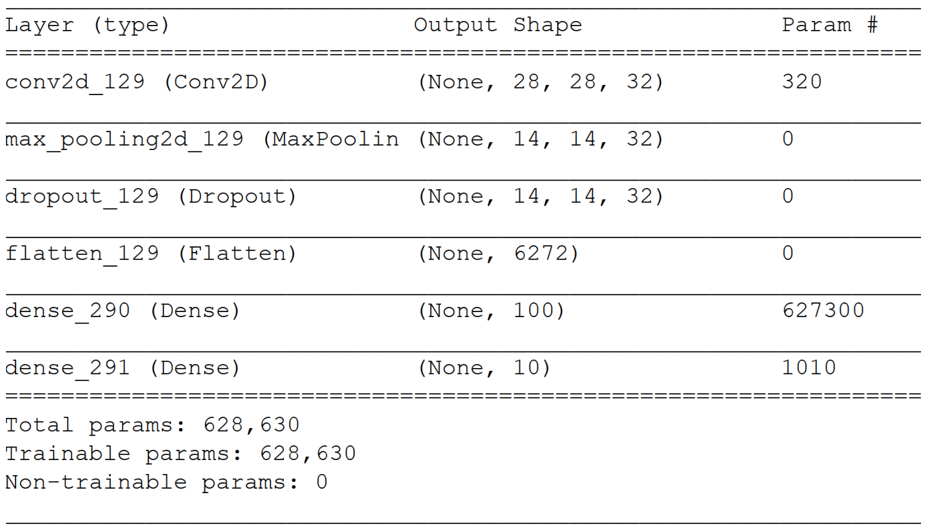

Some of the wrong predictions in cnn_kf model:
Predicted Shirt(6), Class Pullover(2)
Predicted Coat(4), Class Shirt(6)
Predicted Coat(4), Class Shirt(6)
Predicted Coat(4), Class Pullover(2)
Predicted T-shirt/top(0), Class Shirt(6)
Predicted Shirt(6), Class T-shirt/top(0)
Predicted Pullover(2), Class Dress(3)
Predicted Shirt(6), Class T-shirt/top(0)

Figure 9:

• Visualizing Convolutional Layers

Finally, let's see what is going on under the hood while we process the images. A cnn models, has two major steps: extracting the features which occurs when the images passing through the series of convolution layers (step one), and classifying the images by flattening the output of the convolutional layers and applying the softmax function to classify objects with probabilistic values between 0 and 1.
Convolution an image with different filters can perform operations such as edge detection, blur and sharpen by applying filters.
As the Fig. **??** shows, the first convolutional layer has 32 channels. Fig. 1 shows how a random image is convoluted in each of these 32 channels.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_129 (Conv2D)          (None, 28, 28, 32)        320
_________________________________________________________________
max_pooling2d_129 (MaxPoolin (None, 14, 14, 32)        0
_________________________________________________________________
dropout_129 (Dropout)        (None, 14, 14, 32)        0
_________________________________________________________________
flatten_129 (Flatten)        (None, 6272)              0
_________________________________________________________________
dense_290 (Dense)            (None, 100)               627300
_________________________________________________________________
dense_291 (Dense)            (None, 10)                1010
=================================================================
Total params: 628,630
Trainable params: 628,630
Non-trainable params: 0
_________________________________________________________________
```

Figure 10: `cnn_1.summary()`

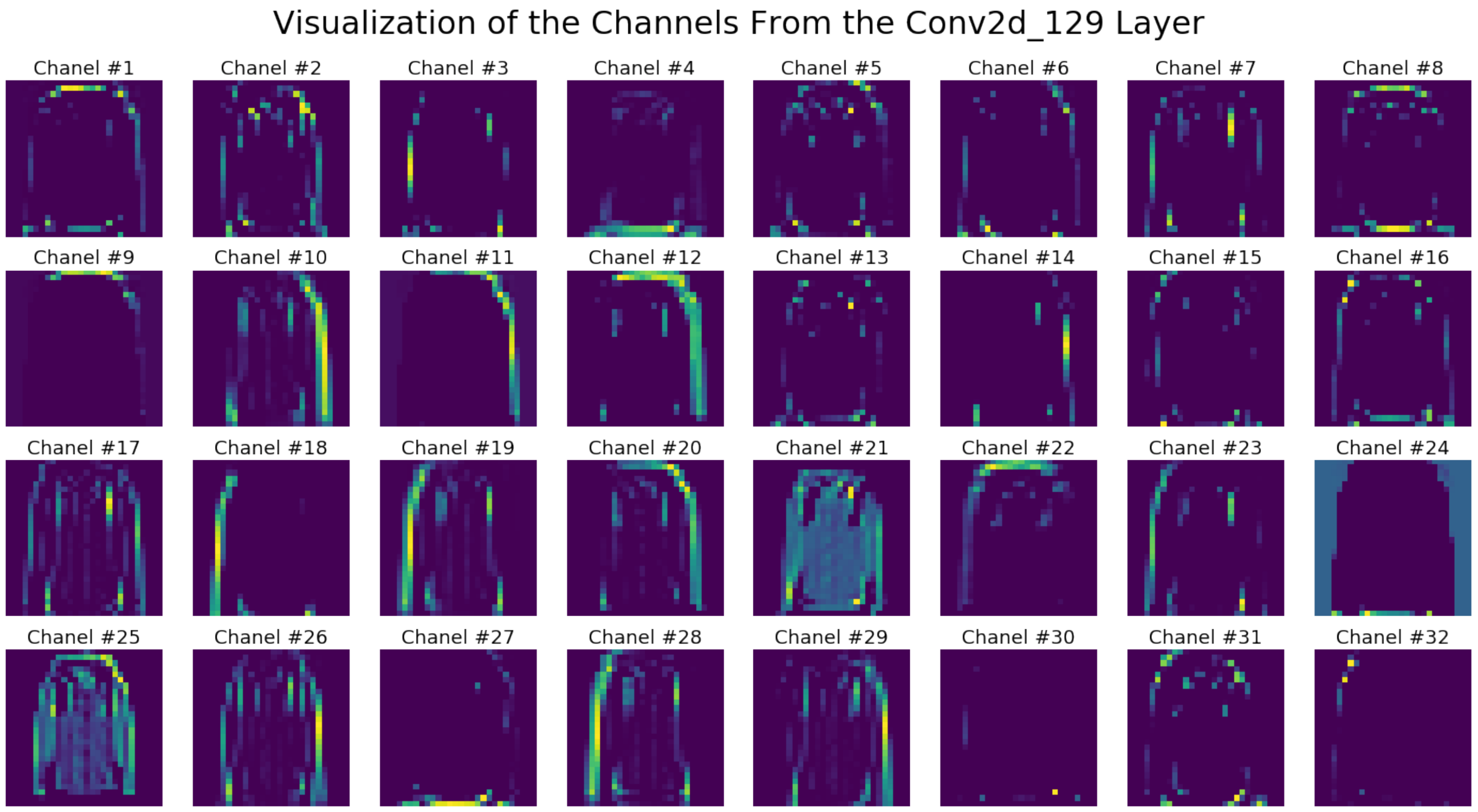

Visualization of the Channels From the Conv2d_129 Layer
Chanel #1
Chanel #2
Chanel #3
Chanel #4
Chanel #5
Chanel #6
Chanel #7
Chanel #8
Chanel #9
Chanel #10
Chanel #11
Chanel #12
Chanel #13
Chanel #14
Chanel #15
Chanel #16
Chanel #17
Chanel #18
Chanel #19
Chanel #20
Chanel #21
Chanel #22
Chanel #23
Chanel #24
Chanel #25
Chanel #26
Chanel #27
Chanel #28
Chanel #29
Chanel #30
Chanel #31
Chanel #32

Figure 11: