Q2 - Classification : Convolutional Neural Networks

Objective :

In this part, we have experimented with Fashion-MNIST dataset using various CNN models. The objective is to identify (predict) different fashion products from the given images using various best possible CNN Models and compare their results (performance measures/scores) to arrive at the best Deep learning model.

Acknowledgements :

We have referred various kaggle notebooks to understand how to approach a machine learning problem and to understand how other people have approached this dataset in the past. These are the few resources we used to refer,

- https://www.kaggle.com/fuzzywizard/fashion-mnist-cnn-keras-accuracy-93
- https://github.com/zalandoresearch/fashion-mnist
- http://athena.ecs.csus.edu/~hoangkh/Image%20Classification%20with%20Fashion-MNIST%20and%20CIFAR-10%20-%20Final%20Report.pdf
- https://www.kaggle.com/zalando-research/fashionmnist

Dataset

Fashion MNIST Training dataset consists of 60,000 images and each image has 784 features (i.e. 28×28 pixels). Each pixel is a value from 0 to 255, describing the pixel intensity. 0 for white and 255 for black.

2.1: Explanation of Design and Implementation Choices of your Model

Based on our learnings from the above references, we learned how CNN works and how to build a simple CNN model. So, for this question, we approached it in the following way.

Step 1- Using Custom CNN model:

- 1 - Convolution layer CNN (Name: Model 1)
- 3 - Convolution layer CNN (Name: Model 3)
- 4 - Convolution layer CNN (Name: Model 4) - ***Best model for us***

Step 2 - Using Transfer Learning:

- ResNet50 pretrained with ImageNet

From the above models, we choose the best one.

We will first split the original training data (60,000 images) into 80% training (48,000 images) and 20% validation (12000 images) optimize the classifier, while keeping the test data (10,000 images) to finally evaluate the accuracy of the model on the data it has never seen. This helps to see whether I'm over-fitting on the training data and whether I should lower the learning rate and train for more epochs if validation accuracy is higher than training accuracy or stop over-training if training accuracy shifts higher than the validation.

Pre-Processing

- Reshape the train and test dataset to 28 X 28 dimension, so that it can be fed as input images to any CNN models.
- Normalization of the data (Feature Scaling)

This data has grayscale images of 28*28 pixels so the pixel can take any value between 0 to 255 and hence we try to normalize or rescale the data to unit dimension so that the value lies between 0-1 which allows the gradient descent to converge faster thereby reducing the training time.

- Split the training data into training and validation set.

Designing the CNN model

This CNN takes as input tensors of shape (image_height, image_width, image_channels). In this case, I configure the CNN to process inputs of size (28, 28, 1), which is the format of the FashionMNIST images. I do this by passing the argument input_shape=(28, 28, 1) to the first layer.

Here, we are explaining how we built the Model 4 as it is the best one we had. The steps are similar to other models.

Model 4: 4 - Convolution layered CNN

- The 1st layer is a Conv2D layer for the **convolution** operation that extracts features from the input images by sliding a convolution filter over the input to produce a feature map. Here we choose a feature map with size 3 x 3.
- The MaxPooling2D layers are used for the **max-pooling** operation that reduces the dimensionality of each feature, which helps shorten training time and reduce number of parameters. Here we choose the pooling window with size 2 x 2.
- To normalize the input layers, we use the **BatchNormalization** layers to adjust and scale the activations. Batch Normalization reduces the amount by what the hidden unit values shift around (covariance shift). Also, it allows each layer of a network to learn by itself a little bit more independently of other layers.
- To combat overfitting, we use the **Dropout** layers, a powerful regularization technique. Dropout is the method used to reduce overfitting. It forces the model to learn multiple independent representations of the same data by randomly disabling neurons in the learning phase. For example, the 1st dropout layer will randomly disable 20% of the outputs.
- In total, this model has 4 Conv2D layers, 2 MaxPooling layers, 6 BatchNormalization layers, and 5 Dropout layers.
- The next step is to feed the last output tensor into a stack of Dense layers, otherwise known as **fully-connected** layers. These densely connected classifiers process vectors, which are 1D, whereas the current output is a 3D tensor. Thus, we need to **flatten** the 3D outputs to 1D, and then add 2 Dense layers on top.
- We do a 5-way classification (as there are 5 classes of fashion images), using a final layer with 5 outputs and a **softmax** activation. Softmax activation enables us to calculate the output based on the probabilities. Each class is assigned a probability and the class with the maximum probability is the model's output for the input.

ResNet50 using Transfer learning

A pretrained network is a saved network that was previously trained on a large dataset, typically on a large-scale image-classification task. If this original dataset is large enough and general enough, then the spatial hierarchy of features learned by the pretrained network can effectively act as a generic model of the visual world, and hence its features can prove useful for many different computer-vision problems, even though these new problems may involve completely different classes than those of the original task.

- The dataset shape we have is 28 X 28 X 1 . we reshape it to 32 X 32 X 3 using padding.
- Here, we use **Imagenet** weights to initialize the model.
- Add bottleneck layers (Dense + BN + Activation + Dropout + Output)
- Froze the base layers of the model
- Trained the bottleneck layers for 5 epochs
- 'Unfroze' the last two resnet blocks
- Re-compiled and re-trained with Adam and a small LR.

The above procedure is given in the Keras documentation for transfer learning.

2.2 Implementation of your Design Choices

Using 1-Convolution layer

```
# Import Keras libraries
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D


# Each image's dimension is 28 x 28
img_rows, img_cols = 28, 28
input_shape = (img_rows, img_cols, 1)

model1 = Sequential()
model1.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=input_shape))
model1.add(MaxPooling2D(pool_size=(2, 2)))
model1.add(Dropout(0.2))

model1.add(Flatten())

model1.add(Dense(64, activation='relu'))
model1.add(Dense(5, activation='softmax'))

model1.compile(loss=keras.losses.categorical_crossentropy,
               optimizer=keras.optimizers.Adam(),
               metrics=['accuracy'])

model1.summary()
```

Using 3-Convolution layer

```
model3 = Sequential()
model3.add(Conv2D(32, kernel_size=(3, 3), activation='relu',input_shape=input_shape))
model3.add(MaxPooling2D((2, 2)))
model3.add(Dropout(0.2))

model3.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model3.add(MaxPooling2D(pool_size=(2, 2)))
model3.add(Dropout(0.2))

model3.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model3.add(Dropout(0.4))

model3.add(Flatten())

model3.add(Dense(128, activation='relu'))
model3.add(Dense(5, activation='softmax'))

model3.compile(loss=keras.losses.categorical_crossentropy,
               optimizer=keras.optimizers.Adam(),
               metrics=['accuracy'])

model3.summary()
```

Using 4-Convolution layers

```
model4 = Sequential()
model4.add(Conv2D(32, kernel_size=(3, 3), padding='same',activation='relu', input_shape=(28,28,1)))
model4.add(BatchNormalization())

model4.add(Conv2D(32, kernel_size=(3, 3), padding='same',activation='relu'))
model4.add(BatchNormalization())
model4.add(MaxPooling2D(pool_size=(2, 2)))
model4.add(Dropout(0.2))

model4.add(Conv2D(64, kernel_size=(3, 3), padding='same',activation='relu'))
model4.add(BatchNormalization())
model4.add(Dropout(0.2))

model4.add(Conv2D(128, kernel_size=(3, 3),padding='same',activation='relu'))
model4.add(BatchNormalization())
model4.add(MaxPooling2D(pool_size=(2, 2)))
model4.add(Dropout(0.3))

model4.add(Flatten())

model4.add(Dense(512, activation='relu'))
model4.add(BatchNormalization())
model4.add(Dropout(0.4))

model4.add(Dense(128, activation='relu'))
model4.add(BatchNormalization())
model4.add(Dropout(0.4))

model4.add(Dense(5, activation='softmax'))

model4.compile(loss=keras.losses.categorical_crossentropy,
               optimizer=keras.optimizers.Adam(lr=0.001),
               metrics=['accuracy'])
```

```
import tensorflow as tf
from keras.models import Model
from keras.applications.resnet50 import ResNet50
from keras.layers import Dense, GlobalAveragePooling2D, Flatten

resnet_model=ResNet50(weights='imagenet', include_top=False, input_shape=(32,32,3))

model=Sequential()
model.add(resnet_model)
model.add(GlobalAveragePooling2D())
model.add(Dense(256, activation='relu'))
model.add(Dropout(.3))
model.add(BatchNormalization())
model.add(Dense(5, activation='softmax'))
for layer in resnet_model.layers:
    if isinstance(layer, BatchNormalization):
        layer.trainable = True
    else:
        layer.trainable = False


model.summary()
```

2.3 Kaggle Competition Score

The Highest score in Kaggle for question 2 - ***90.64% Test Accuracy***. And, the corresponding model and parameters used are,

Classification using CNN model : *4-layered custom CNN (Refer 2.1 for details)*

Though it is a good score for a classification model, some of our peers were able to achieve more than 91% accuracy using deep learning approach. Following could be the reason for their accuracies,

1. We can use Model Pruning with proper optimization. This could help us remove some unwanted weights and layers in the model.
2. Proper fine tuning after transfer learning of pretrained models. We were not able to understand a certain approach in the given time.
3. Proper data augmentation to increase the training dataset, so that the model can absorb the features better.

2.4 Results Analysis

1) Runtime performance for training and testing.

Model 1

Epoch 50/50

48000/48000 [==============================] - 2s 34us/step - loss: 0.1417 - accuracy: 0.9449 - val_loss: 0.3379 - val_accuracy: 0.8834

time: 1min 29s

Model 3

Epoch 50/50

48000/48000 [==============================] - 2s 44us/step - loss: 0.2384 - accuracy: 0.9027 - val_loss: 0.2558 - val_accuracy: 0.8984

time: 1min 48s

Model 4

Epoch 30/30

48000/48000 [==============================] - 7s 149us/step - loss: 0.1894 - accuracy: 0.9246 - val_loss: 0.2466 - val_accuracy: 0.9061

time: 3min 40s

ResNet50

Epoch 400/400

48000/48000 [==============================] - 10s 203us/step - loss: 0.3266 - accuracy: 0.8658 - val_loss: 0.4189 - val_accuracy: 0.8342

time: 1h 4min 3s

2) Comparison of the different algorithms and parameters you tried.

Model	Epoch / Batchsize	Time (mins)	Train accuracy	Validation Accuracy	Test Accuracy	Train Loss	Validation Loss
Model 1	50 - 512	1.29	0.9449	0.8834	0.8412	0.1417	0.3379
Model 3	50 - 256	1.48	0.9027	0.8984	0.8892	0.2384	0.2558
Model 4	**30 - 256**	**3.40**	**0.9246**	**0.9061**	**0.9028**	**0.1894**	**0.2466**
ResNet50	400 - 256	64.3	0.8658	0.8571	0.8342	0.3266	0.418

3) Explanation of your model (algorithms, network architecture, optimizers, regularization, design choices, numbers of parameters)

Model	Architecture	Training Accuracy	Validation Accuracy	Test Accuracy
Model 4	4 convolution layers 2 max.pool layers 5 Dropout layers 6 Batch Normalization 2 fully-connected layers Optimizer - Adam(lr=0.001) Loss - categorical_crossentropy	0.9246	0.9061	0.9028

Model 4: 4 - Convolution layered CNN

- The 1st layer is a Conv2D layer for the convolution operation that extracts features from the input images by sliding a convolution filter over the input to produce a feature map. Here we choose a feature map with size 3 x 3.
- The MaxPooling2D layers are used for the max-pooling operation that reduces the dimensionality of each feature, which helps shorten training time and reduce number of parameters. Here we choose the pooling window with size 2 x 2.
- To normalize the input layers, we use the BatchNormalization layers to adjust and scale the activations. Batch Normalization reduces the amount by what the hidden unit values

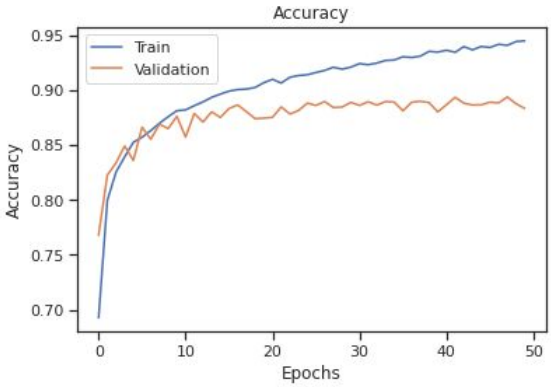
shift around (covariance shift). Also, it allows each layer of a network to learn by itself a little bit more independently of other layers.

- To combat overfitting, we use the Dropout layers, a powerful regularization technique. Dropout is the method used to reduce overfitting. It forces the model to learn multiple independent representations of the same data by randomly disabling neurons in the learning phase. For example, the 1st dropout layer will randomly disable 20% of the outputs.
- In total, this model has 4 Conv2D layers, 2 MaxPooling layers, 6 BatchNormalization layers, and 5 Dropout layers.
- The next step is to feed the last output tensor into a stack of Dense layers, otherwise known as fully-connected layers. These densely connected classifiers process vectors, which are 1D, whereas the current output is a 3D tensor. Thus, we need to flatten the 3D outputs to 1D, and then add 2 Dense layers on top.
- We do a 5-way classification (as there are 5 classes of fashion images), using a final layer with 5 outputs and a softmax activation. Softmax activation enables us to calculate the output based on the probabilities. Each class is assigned a probability and the class with the maximum probability is the model's output for the input.

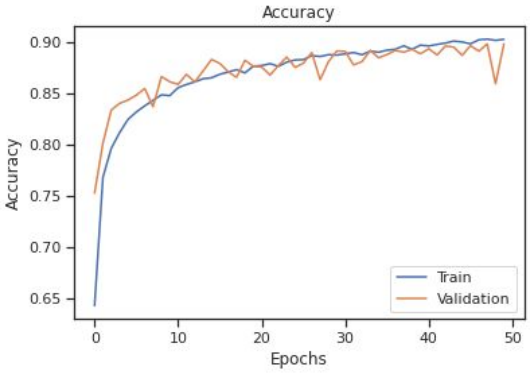
4) **You can use any plots to explain the performance of your approach. But at the very least produce two plots, one of training epoch vs. loss and one of classification accuracy vs. loss on both your training and test set.**

Accuracy vs Epochs plots:

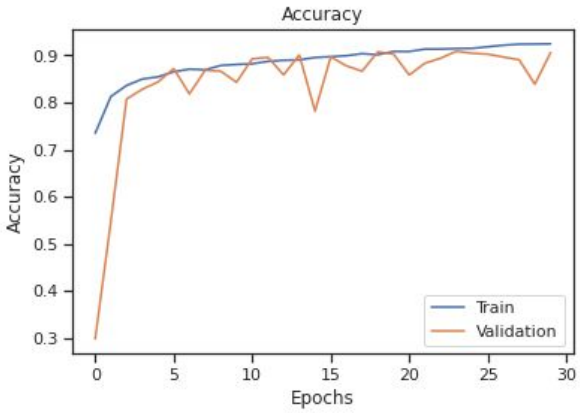
Model 1



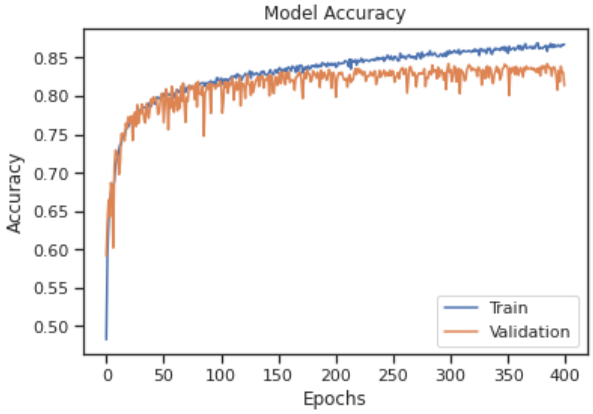
Model 3



Model 4

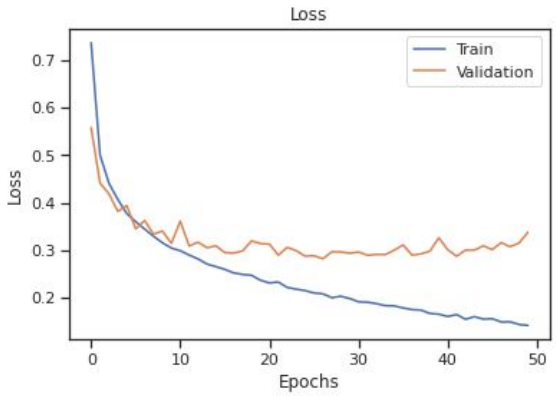


ResNet50

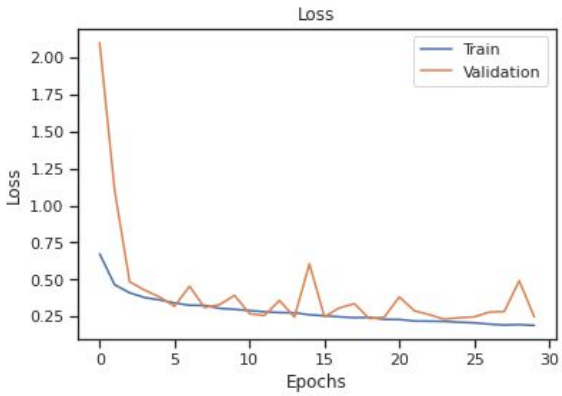


Loss vs Epochs:

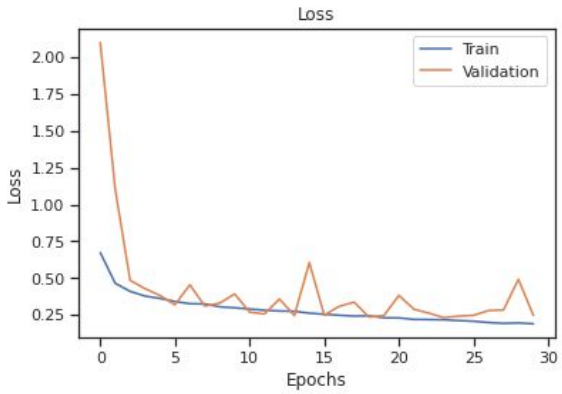
Model 1



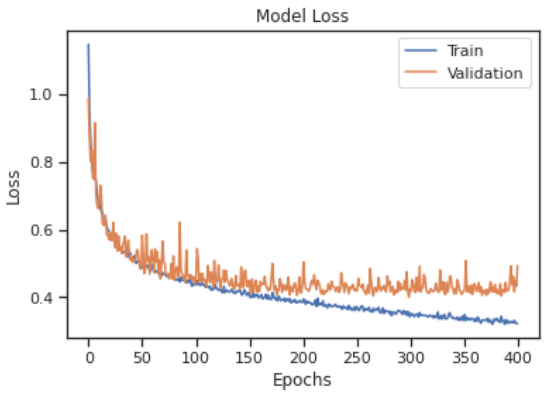
Model 3



Model 4



ResNet50



Key points:

- We can see in model 1, it is clearly overfitting from accuracy and loss plot.
- Model 3 seems to be fine without overfitting, but accuracy is less.
- Model 4 is good, as there is no overfitting and also it has better accuracy.
- Through ResNet50 with pre-trained ‘Imagenet’ weights. We can see it also overfits.
- The fluctuations in the plot of Model 1, Model 3, Model 4 is due to less train data.

5) **Evaluate your code with other metrics on the training data (by using some of it as test data) and argue for the benefit of your approach.**

	precision	recall	f1-score	support
Class 0	0.92	**0.99**	**0.95**	1568
Class 1	0.83	0.89	0.86	1446
Class 2	0.85	0.83	0.84	**1572**
Class 3	0.90	0.84	0.87	1567
Class 4	**0.96**	0.91	0.94	1497

From the above table we can see,

- Class 4 has highest precision
- Class 0 has highest recall
- Class 2 has highest support
- Class 0 has the highest f1-score.

More Precision indicates that an example which is classified as positive is indeed positive. From the table it is also observed that, if we have a high precision, then we have a lower recall and if we have low precision, we have a higher recall. High Precision, low recall indicates that we did miss a lot of Positives and there are a lot of False Negatives. But the ones classified as Positive are indeed positive as there are less False Negatives. Low Precision, High Recall indicates that there are a lot of False Positives, but the positive examples are correctly recognized as there are very less False Negatives. Class 4 examples classified as positives are classified as positive correctly is maximum among the four classes, while it is least for Class 0.

F1-score is a measure of the test's accuracy. It has a maximum value of 1. Higher value of f1-score means that balance between the precision and recall is more. Class 0 has a better balance between precision and recall than all other classes.