CA1 - Image Classification on FashionMNIST

Name: Quah Johnnie
Class: DAAA/FT/2B/04
Admin No: 2007476

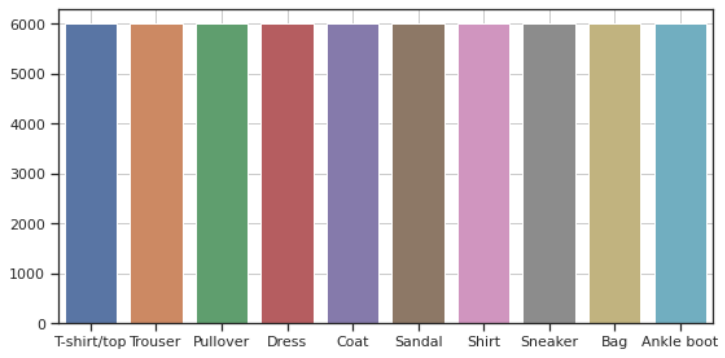
Contents (Jupyter Highlights)

Sections	Remarks
1.0) Dataset Preparation & EDA	Distribution, Glance of Dataset, Outliers
2.0) Data Augmentation	Data splitting & Augmentation
3.0) Modelling	Types of model used & Evaluation
4.0) Model Improvement	Comparing Augmentation & Coarse To Fine Hyperparameter Tuning
5.0) Final Evaluation	Test Evaluation & Feature Mapping (1st layer)
Conclusion	Conclusion & Rooms for possible improvement

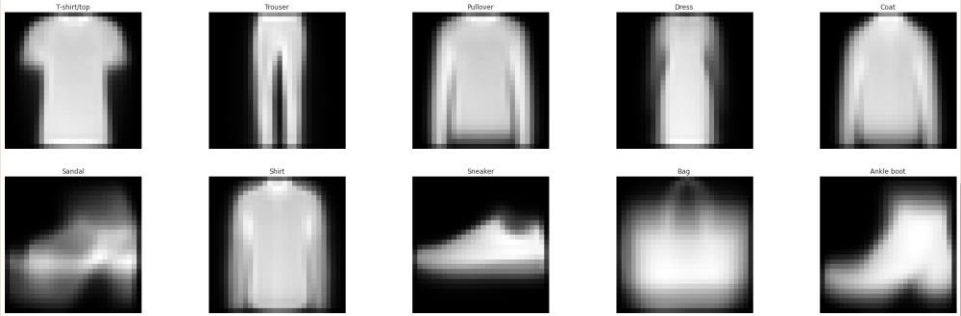
Current Highest Validation Accuracy: NIL

1.0) FashionMNIST at a glance (EDA)

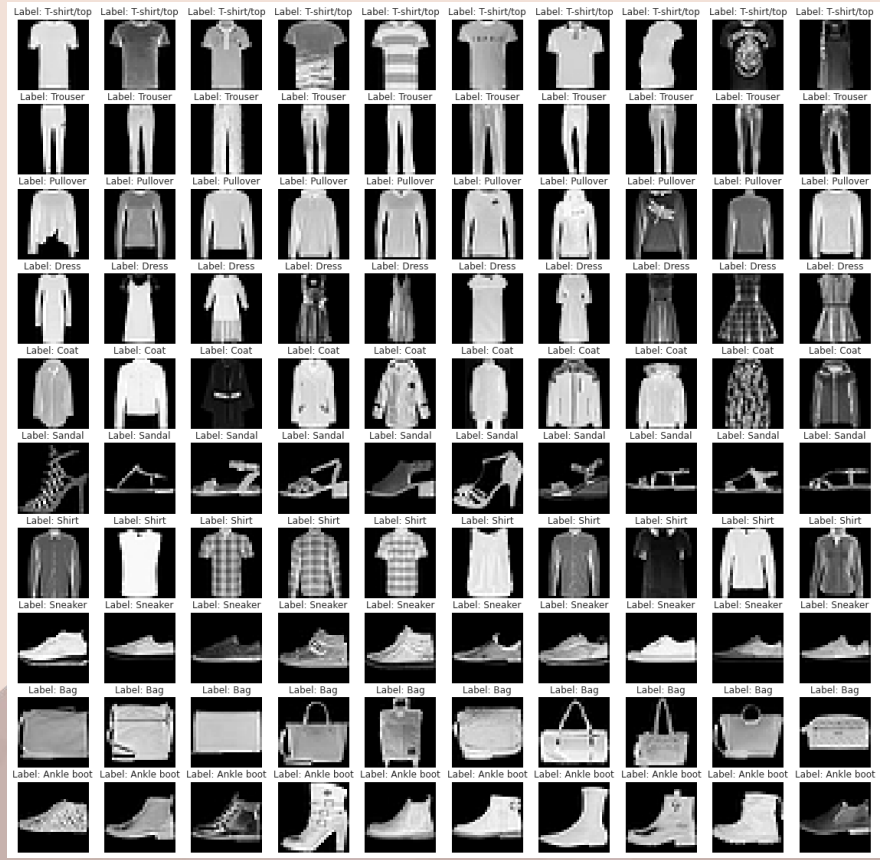
Training data distribution



Average image of each class

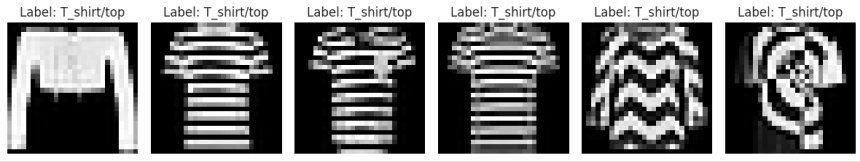


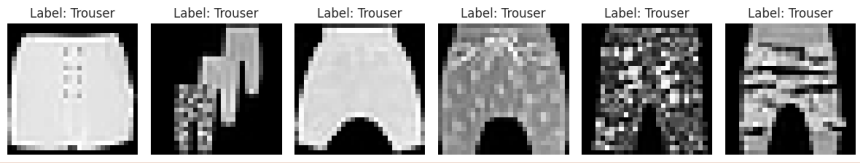
Glance of each class image

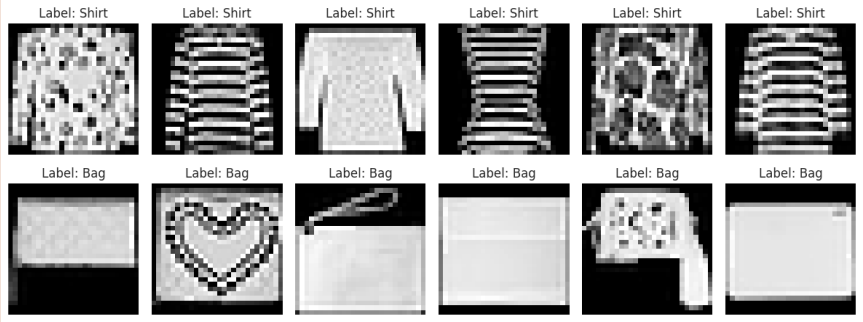


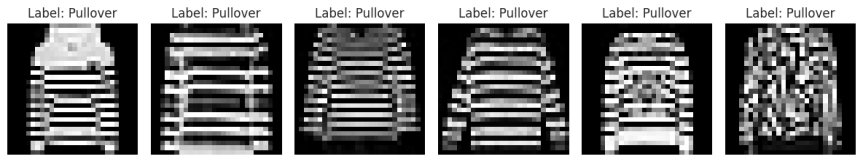
Current Highest Validation Accuracy: NIL

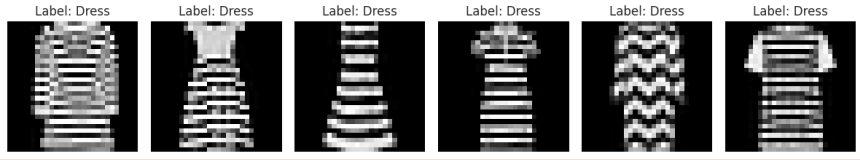
1.0) Outliers Using Autoencoder

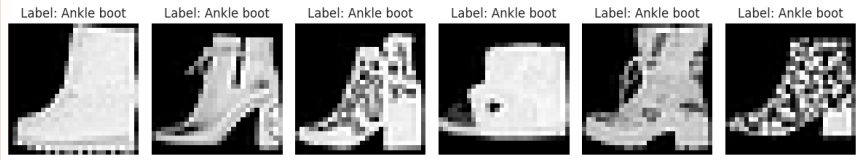


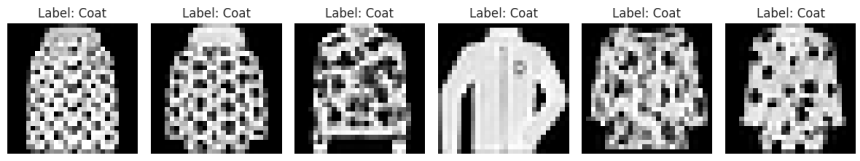


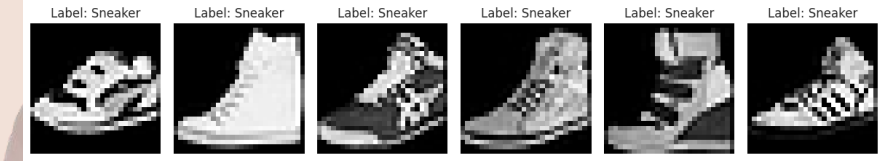












Current Highest Validation Accuracy: NIL

2.0) Data Preparation & Augmentation

DATA SPLIT

Training Data split into:

- 40K Training
- 10K Validation
- 10K Testing

DATA AUGMENTATION

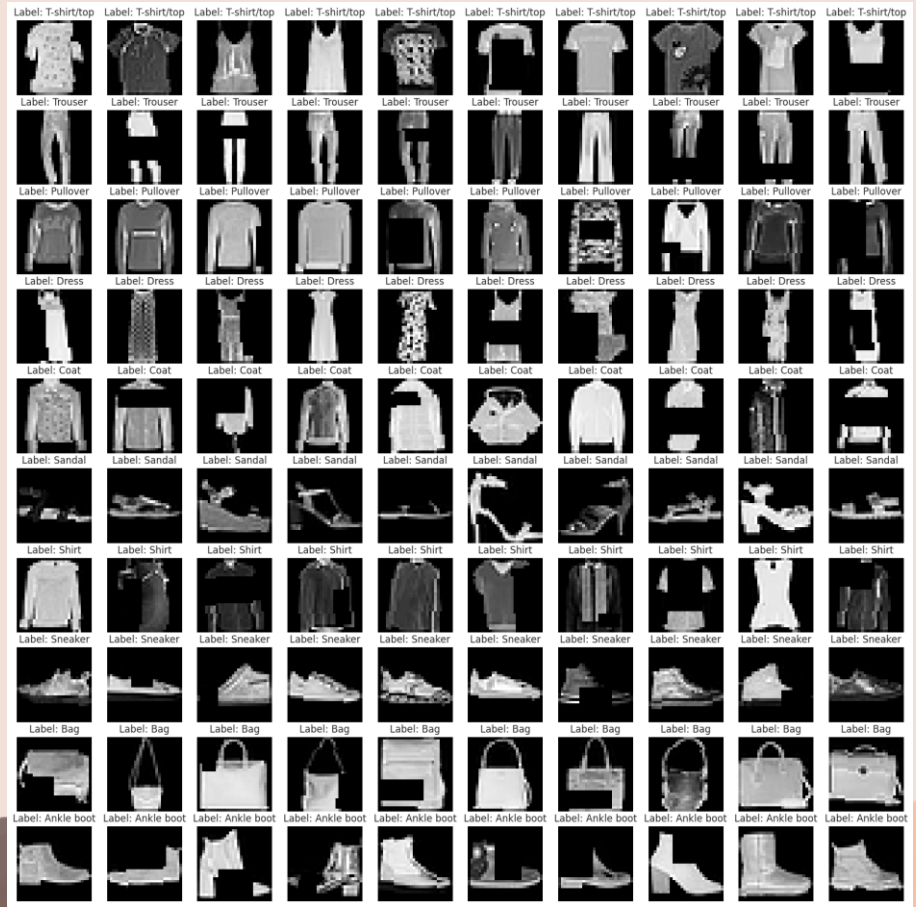
Composing Augmentation Sets With Combinations of:

- RandomRotation(5°)
- RandomHorizontalFlip(0.8)
- RandomPerspective(0.115,0.4)
- Replication

Current Highest Validation Accuracy: NIL

2.0) Visualization of Augmentation Used

RandomErasing seen
HorizontalFlip seen
Slight Rotation seen
Slight Distortion seen



Current Highest Validation Accuracy: 94.68%

3.0) Modelling - Model architectures tried

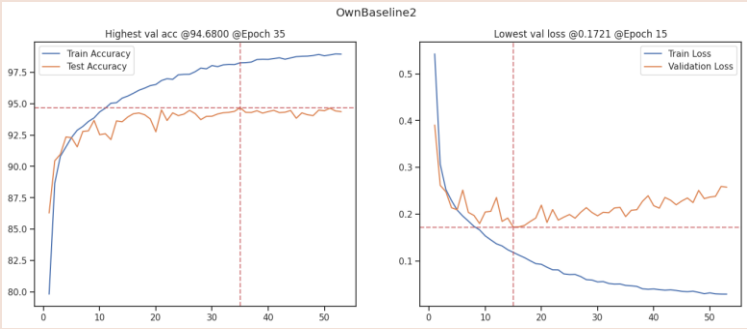
Model	Layers (Conv+FC)	Parameters
OwnBaseline1	7	937,834
OwnBaseline2	9	2,917,482
OwnBaseline3	16	4,801,578
ResNet4Blocks	14	8,028,618
ResNet3Blocks	14	2,258,890
DenseNet4Blocks	93	1,253,276
DenseNet3Blocks	93	1,122,830

VGGNet inspired

ResNet architecture used

DenseNet architecture used

Best model (VGGNet inspired)



Overfitting seen

Lowest Val Loss	Highest Val Acc	Lowest Train Loss	Highest Train Acc	Epoch (Highest Val Acc)	Model Description	Parameter
0.179700	94.32	0.054980	98.004	35	OwnBaseline1	937,834
0.172123	94.68	0.028777	98.968	35	OwnBaseline2	2,917,482
0.245260	92.57	0.053536	98.206	40	OwnBaseline3	4,801,578
0.249656	91.33	0.061010	97.724	22	ResNet4Blocks	8,028,618
0.253545	91.69	0.007266	99.740	50	ResNet3Blocks	2,258,890
0.305238	90.11	0.014231	99.514	36	DenseNet4Blocks	1,253,276
0.295854	89.87	0.039509	98.612	10	DenseNet3Blocks	1,122,830
0.305238	90.11	0.014231	99.514	36	DenseNet4Blocks	1,253,276
0.295854	89.87	0.039509	98.612	10	DenseNet3Blocks	1,122,830

Trained with original data.

Current Highest Validation Accuracy: 95.17% (+0.49%)

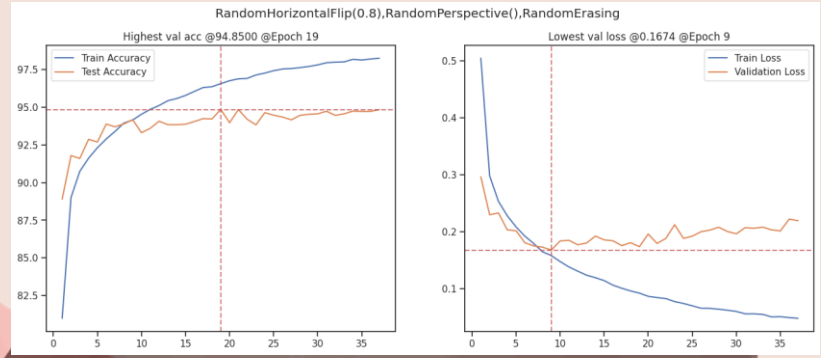
4.0) Model Improvement - Tested different types of augmentation

Split	Size	Description
Trainset 1	50K	Normalization (min-max)
Trainset 2 (Replicated Data)	50k + 50k	Normalization (min-max), RandomHorizontalFlip(0.8), RandomErasing()
Trainset 3 (Replicated Data)	50k + 50k	Normalization (min-max), RandomHorizontalFlip(0.8), RandomRotation(-5,5)
Trainset 4 (Replicated Data)	50k + 50k	Normalization (min-max), RandomHorizontalFlip(0.8), RandomErasing(), RandomRotation(5)
Trainset 5 (Replicated Data)	50k + 50k	Normalization (min-max), RandomErasing(), RandomRotation(-5,5)
Trainset 6 (Replicated Data)	50K + 50k	Normalization (min-max), RandomHorizontalFlip(0.8), RandomPerspective(0.115,0.4), RandomErasing()

Different types of augmentation sets tried

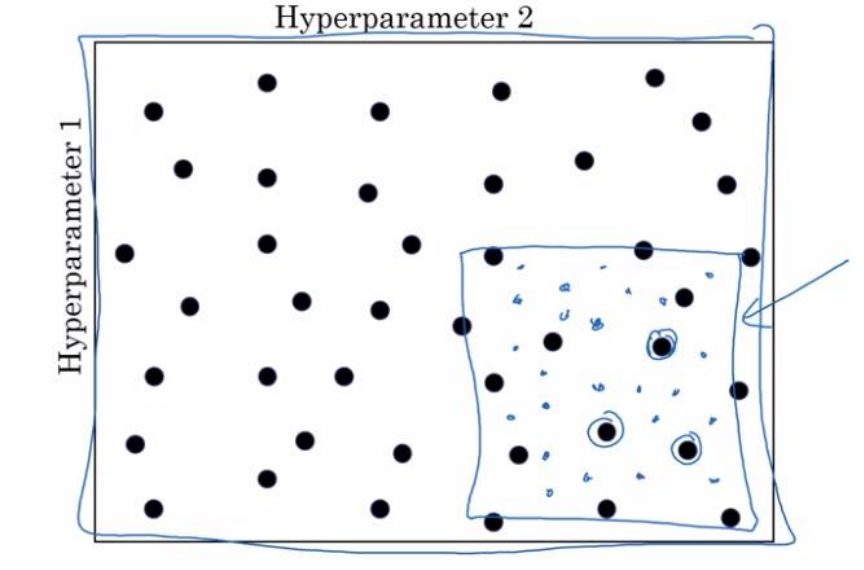
Split	Size	Description	Average Val Acc
Trainset 1	50K	Normalization (min-max)	94.61%
Trainset 3 (Replicated Data)	50k + 50k	Normalization (min-max), RandomHorizontalFlip(0.8), RandomRotation(-5,5)	95.165%
Trainset 4 (Replicated Data)	50k + 50k	Normalization (min-max), RandomHorizontalFlip(0.8), RandomErasing(), RandomRotation(5)	95.17%

The 2 augmentation sets that provided the highest validation accuracy



Current Highest Validation Accuracy: 95.17%

4.0) Model Improvement - Hyperparameter tuning with Coarse To Fine technique



Essentially a double random search, where during the first grid search it will cut down a range of values for my second random search, based on the top 3-5 best hyperparameters (on the first search).

Why this method?

- More efficient way than the traditional random grid search.
- Rather simple dataset; Takes only about 10 minutes to fully train my models (2x 3080ti on PyTorch)

Current Highest Validation Accuracy: 95.42% (+0.25%)

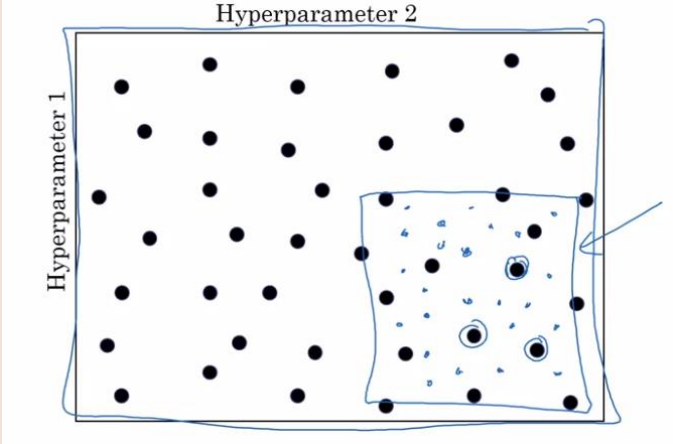
4.0) Model Improvement

Hyperparameters tuned:

- Learning Rate
- Weight Decay
- Momentum
- Layer (No. of fc layers)
- Optimizer (SAM vs SGD)

Batch size at 512

Early Stopper with patience = 18



Random Search 1

```
Trial #75 Finished - Search Time 10.59 Mins
Total Time Elapsed: 917.76 Mins

Hyperparameters         |Trial Values: #75      |Best Trial Values: #67

Learning Rate           |0.011130               |0.005166
Weight Decay (L2)       |0.035938               |0.004642
Momentum                |0.936000               |0.948000
Last Layer No           |3                      |2
Optimizer               |SGD                    |SAM

Highest Val Acc         |94.33                  |95.38
Epoch (Highest Val)     |67                     |64


[0.02397937001275764, 0.03593813663804626, 0.924, 2, 'SGD']


Trial ended at trial #75
```

Range of hyperparameters reduced

```
Trial #70 Finished - Search Time 2.30 Mins
Total Time Elapsed: 935.70 Mins

Hyperparameters         |Trial Values: #70      |Best Trial Values: #9

Learning Rate           |0.040000               |0.014987
Weight Decay (L2)       |0.008000               |0.000619
Momentum                |0.951000               |0.960000
Highest Val Acc         |91.64                  |95.42
Epoch (Highest Val)     |15                     |65


Next trial:  [0.011130237608828497, 0.0017235477520255059, 0.951]


Trial ended at trial #70
```

Test Accuracy: 94.93% (-0.49%)

5.0) Final Evaluation - Error Analysis

Hyperparameters tuned:

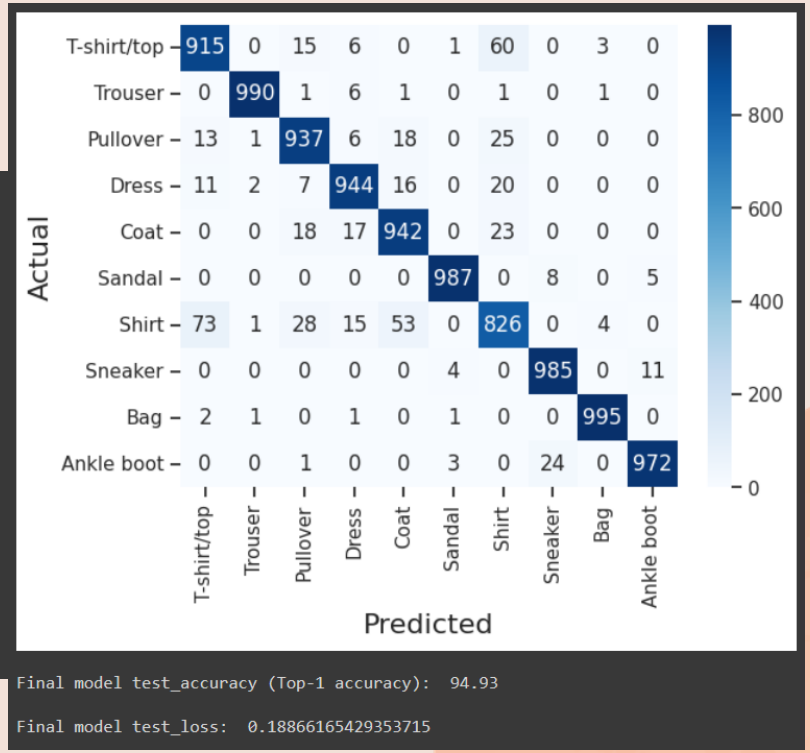
- Fitting the rest of my data, including validation data.
- Hyperparameters and epoch will be based on my best hyperparameters I get from tuning.
- Evaluation on my test set once only, to prevent overfitting on test set.

Highest f1-scores	Class	Actual/Predicted
0.99+	Bag	995/1000
0.99+	Trouser	990/1000
0.99+	Sandal	987/1000

Lowest f1-scores	Class	Actual/Predicted
0.85	Shirt	826/1000
0.91	T_shirt	915/1000
0.93	Coat	942/1000

Final Model

	precision	recall	f1-score	support
T-shirt/top	0.90	0.92	0.91	1000
Trouser	0.99	0.99	0.99	1000
Pullover	0.93	0.94	0.93	1000
Dress	0.95	0.94	0.95	1000
Coat	0.91	0.94	0.93	1000
Sandal	0.99	0.99	0.99	1000
Shirt	0.86	0.83	0.85	1000
Sneaker	0.97	0.98	0.98	1000
Bag	0.99	0.99	0.99	1000
Ankle boot	0.98	0.97	0.98	1000
accuracy			0.95	10000
macro avg	0.95	0.95	0.95	10000
weighted avg	0.95	0.95	0.95	10000



Final model test_accuracy (Top-1 accuracy): 94.93

Final model test_loss: 0.18866165429353715

Test Accuracy: 94.93%

5.0) Final Evaluation – Error Analysis

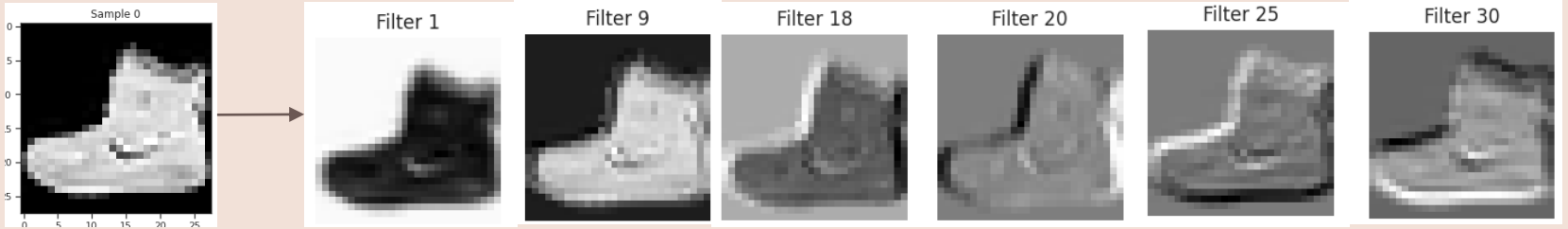
Some mistakes can be forgiven?

Test Accuracy: 94.93%

5.0) Final Evaluation - Feature Mapping (1st layer)

There is 32 filters, but ill present what seems interesting

Original: Ankle Boot



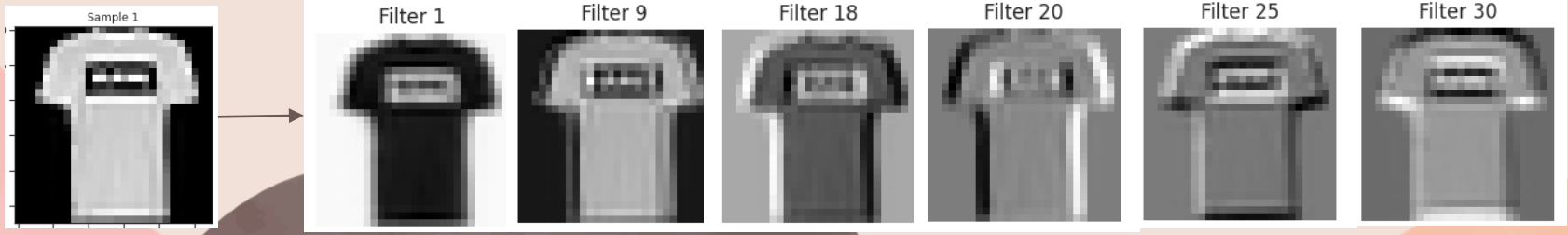
More activated left side

More activated right side

More activated on the top

More activated on the bottom

Original: T-shirt/top



Conclusion

Quick Conclusion:

We did EDA, then splitted our training data into train and validation data. Tried many models before settling on a custom made model similar to VGGNet. We tested multiple augmentation configurations before doing hyperparameter tuning to improve our model and finally did our final evaluation and analysis on our model.

Rooms for possible improvement:

I would say would be to use a smaller batch size. My final model used 512. Perhaps, experimenting with 32/64 would yield a better result. Augmenting my data every epoch might yield a better result, but since this is a relative simple dataset, too much augmentation on training set might harm generalization.

Essential young people memes

Thanks !

Fin. 完。