FASHION MNIST

By Joel Poah
P2112729
DAAA/2B/06

EDA & preprocessing


```
print(X_train.min(), X_train.ma
# it seems like the data set pr
X_train = X_train / 255.0
X_test = X_test / 255.0
X_val = X_val / 255.0
```

```
0 255
```

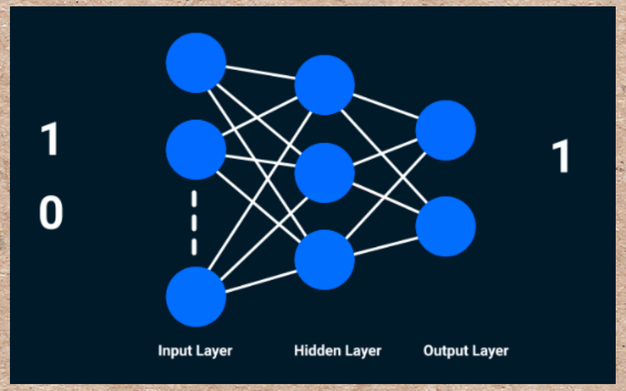
```
[0 1 2 3 4 5 6 7 8 9]
5    4800
4    4800
3    4800
7    4800
2    4800
9    4800
8    4800
0    4800
1    4800
6    4800
Name: 0, dtype: int64
```

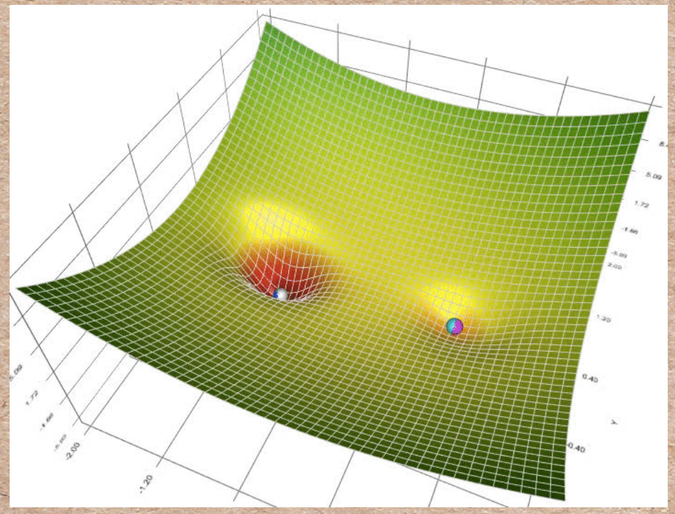
```
stratify=y_train)
```

- Already seeing some similarities even though some pictures are from different classes
- pictures are generally zoomed and centered already (will not zoom for data aug later)
- pixel normalization (dividing by 255 helps keep pixels in a range of 0 to 1 speeds up model else numbers can become very large, other methods include standardization and centering)
- made sure that training set had equal data points for each class using stratified splitting(4800 for each class)

MLP

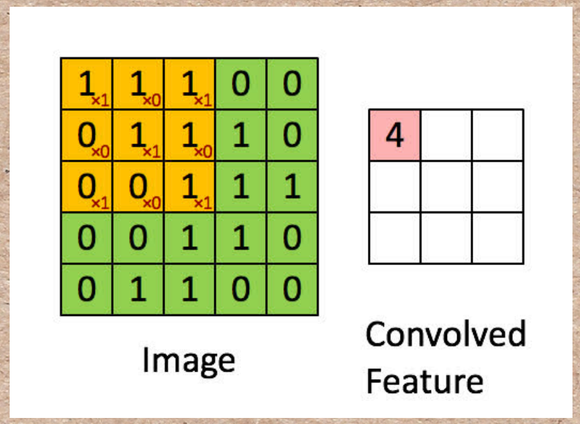
```
X_train = X_train.reshape(-1, 784)
X_test = X_test.reshape(-1, 784)
X_val = X_val.reshape(-1, 784)

y_test_labels = y_test
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
y_val = to_categorical(y_val)
```

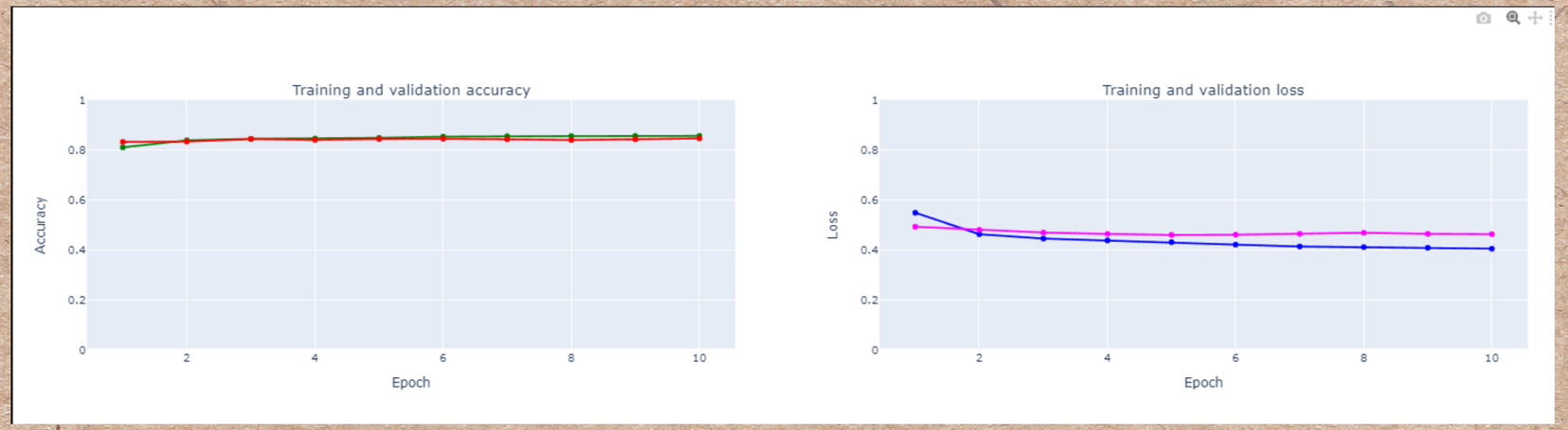




- Just started trying out NN
- flattened array of pixels from 28,28,1 to 784,1
- loss='sparse_categorical_crossentropy" is used for label encoded data
- will be using 'categorical_crossentropy' for one hot encoded data
- First touch on topic of back propagation / gradient descent minimising loss
- played around with 2 - 3 layers was good with 128 neurons beginning then decreasing in nominations of 32 neurons
- 89% to 90% val accuracy pretty good even without CNN

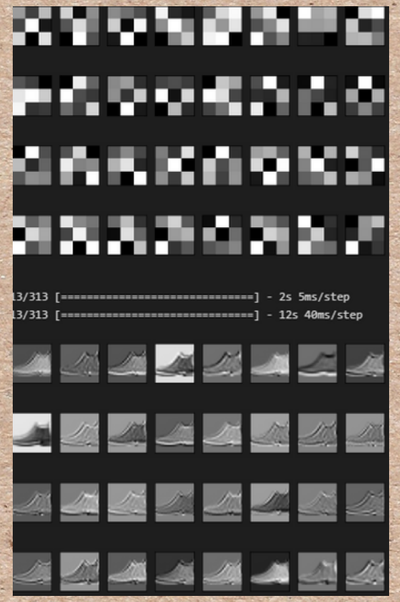
CONV NEURAL NETWORK (CNN)



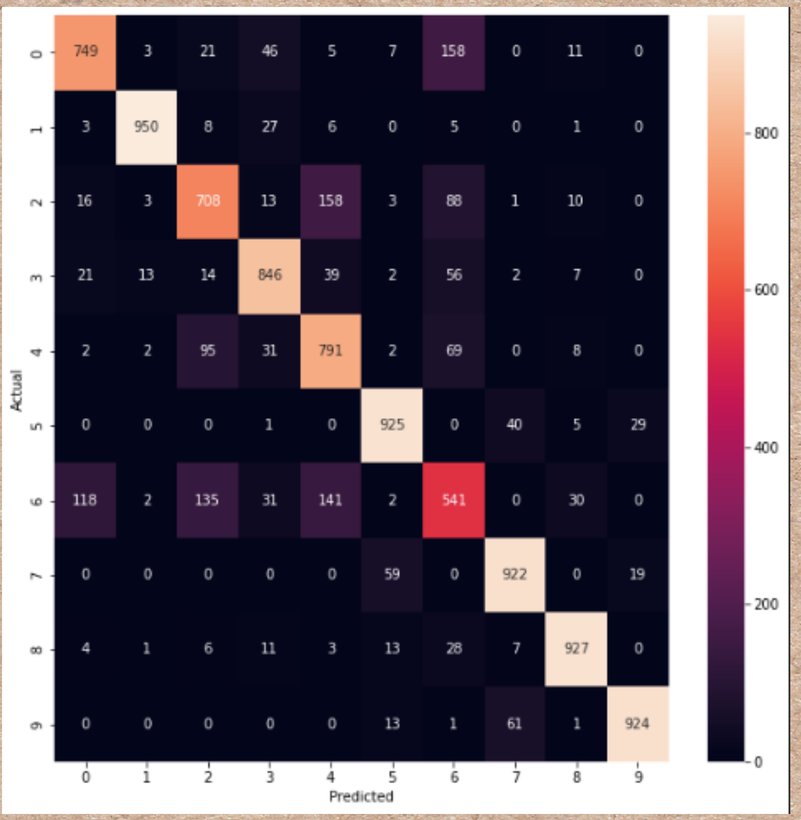


- DOT PRODUCT PERFORMED BETWEEN INPUT AND FILTER(FILTER IS A MEASURE OF HOW CLOSE A REGION OF THE INPUT RESEMBLES A FEATURE: IT CAN BE A VERTICAL LINE,HORIZONTAL EDGE OR DIAGONAL)
- THERE ARE VALUES FOUND IN FILTER(. *FILTER IS ALSO KNOWN AS THE KERNEL_SIZE PARAMETER IN KERAS*) WHICH ARE THE WEIGHTS THAT WILL BE LEARNED AND TUNED WITH BACKPROPAGATION(FINDING GLOBAL MINIMUM ERROR OF FUNCTION) IN THE TRAINING OF THE NETWORK

- IN THE BEGINNING I TRIED CNN WITH ONLY FILTERING AND LINEAR ACTIVATION BASELINE (NO POOLING)
- WILL CONTINUE MORE HYPOTHESIS TEST WITH LINEAR ACTIVATION FOR NOW
- DECENT 84% VAL ACCURACY EVEN W JUST LINEAR ACTIVATION
- ON RIGHT SHOWS FILTERS OF FIRST LAYER
- SHOWS FEATURE MAPS OF A SNEAKER
- SHADES OF FILTERS ARE TRAINED WEIGHTS TO SCAN FOR MEANINGFUL FEATURES
- CAN SEE HOW THE FEATURES MAPS ARE DRAWING OUT THE OUT LINE OF THE SNEAKER

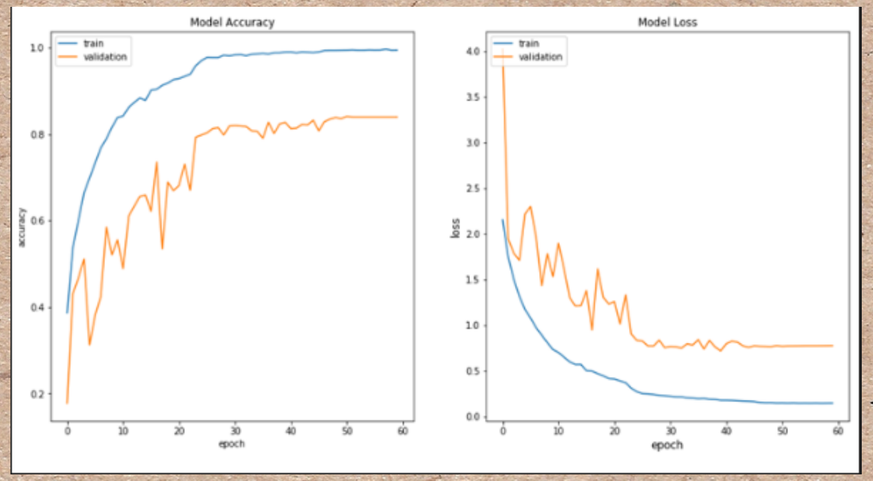


CONV NEURAL NETWORK (CNN)



- CONFUSED BETWEEN CLASSES 0,2,4 AND 6
- T-SHIRT,PULLOVER,COAT AND SHIRT ARE HARD TO DISTINGUISE
- UNDERSTAND FILTERS AND STRIDES BETTER THROUGH BASELINE

accuracy vs loss



- ACCURACY AND LOSS ARE NOT THE SAME
- ACCURACY IS PERCENTAGE OF CORRECT PREDICTIONS
- LOSS IS HOW CONFIDENT MODEL IS FOR A GIVEN CLASS WHEN PREDICTING. PROBABILTY IS GENERATED BY SIGMOID

```
del model
gc.collect()
tf.keras.backend.clear_session()
```

fix memory leak when running ipynb

Various batch size tested (linear act)

```
CPU times: total: 1min 30s
Wall time: 1min 8s
```

default 32 batch size
82.46 val acc

```
CPU times: total: 50.4 s
Wall time: 37.2 s
```

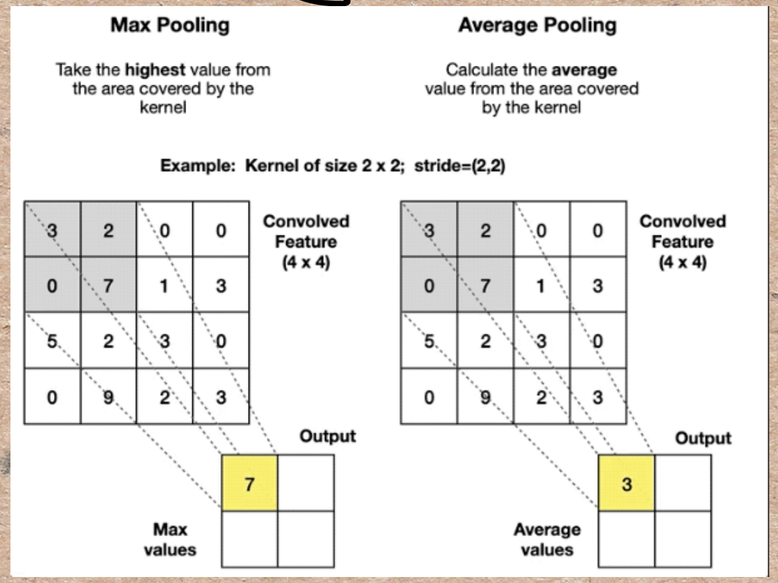
batch_size = 64
82.36 val acc

```
CPU times: total: 31.3 s
Wall time: 23.8 s
```

batch_size = 128
83.02 val acc

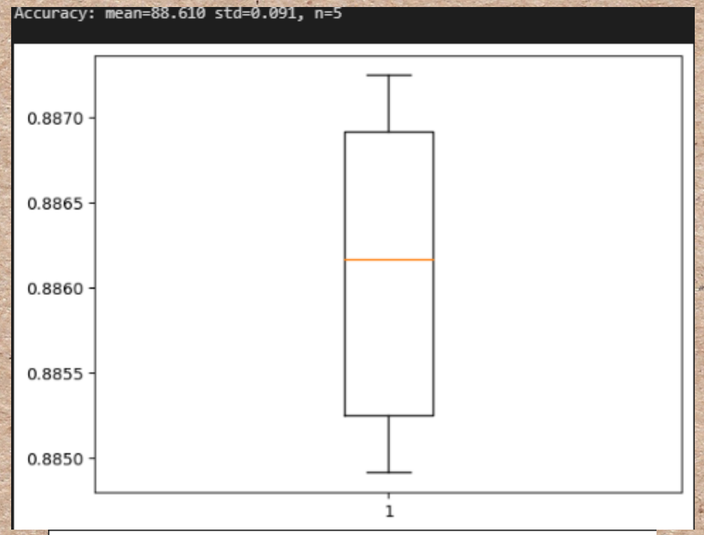
- Not much difference in val accuracy
- 1/3 of compute time needed for 128 batch
- Can afford to run more epochs with 128 batch size

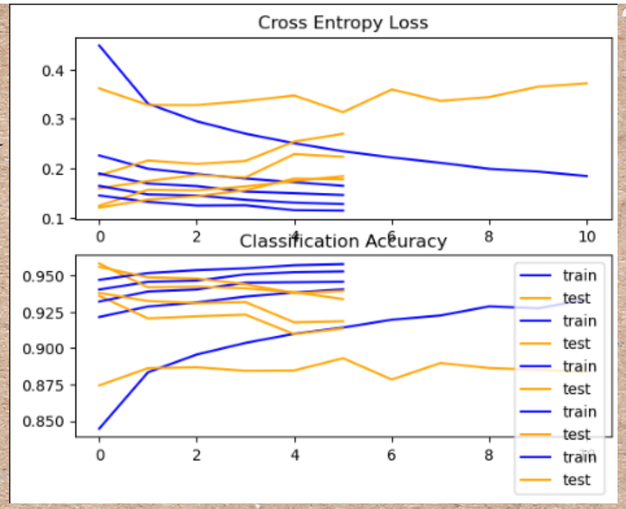
POOLING (CNN)



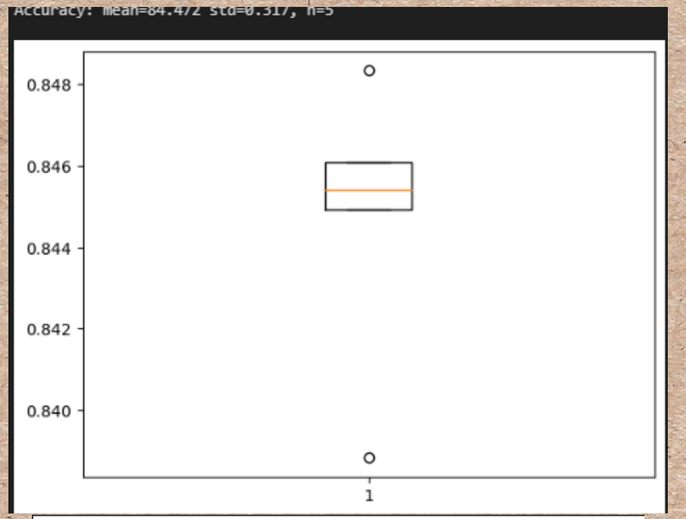
- MAX POOL WORKED BETTER FOR THIS (TAKES THE BRIGHTEST FOR A REGION OF PIXELS)
- AVERAGE POOL TAKES AVERAGE OF THE PIXELS INTENSITY FOR A GIVEN REGION
- MY HYPOTHESIS IS THAT MAX POOLING WILL BE BETTER FOR THIS DATASET AS IT COMPLETELY WIPES OUT NOISE IN A REGION BY TAKING BRIGHTEST PIXEL.
- AS A HUMAN, I WOULD LOOK AT THE BRIGHT EDGES TO DISTINGUISH THE SHAPE/OUTLINE OF THE IMAGE AND DEDUCE WHAT CLASS IT IS. SO THATS WHY I THINK MAX POOLING MIMICS OUR BRAIN FOR THIS PROBLEM MORE AND WORKS BEST HERE

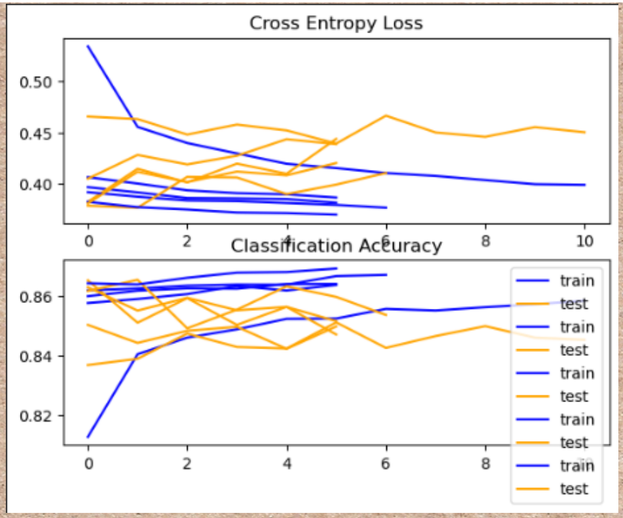
Max 88.6 acc 0.09std





Avg 84.4 acc 0.317 std





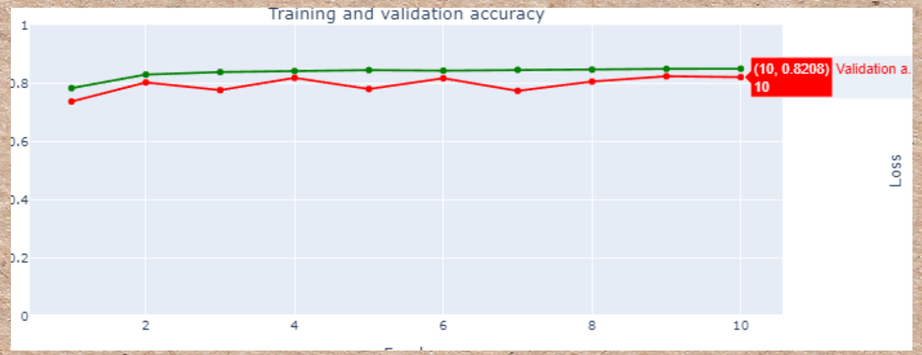
BATCH NORMALIZATION (CNN)

$$X^* = (X - E[X]) / SQRT(VAR(X))$$

where x * is new value of a single component E[x] is mean in a batch var(x) is variance of x in a batch

simple CNN val acc

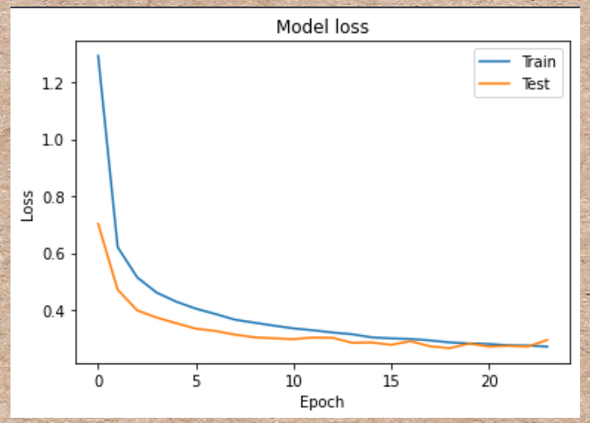
82%

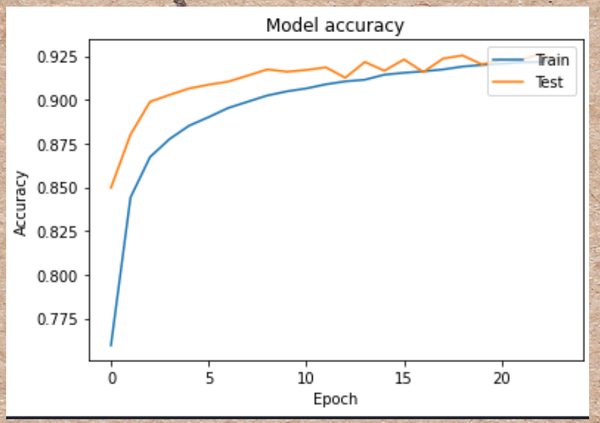


- DIDN'T IMPROVE SCORES AS MUCH AS I THOUGHT
- REALISE IT WORKS BETTER FOR DEEPER NEURAL NETWORKS
- THEORETICALLY HELPS IN INTERNAL COVARIATE SHIFT PROBLEM
- SEEMS TO REGULARIZE THE MODEL NOT OVERFITTING DATA POINTS THAT IS OUT OF THE DISTRIBUTION

deeper CNN val acc

92%





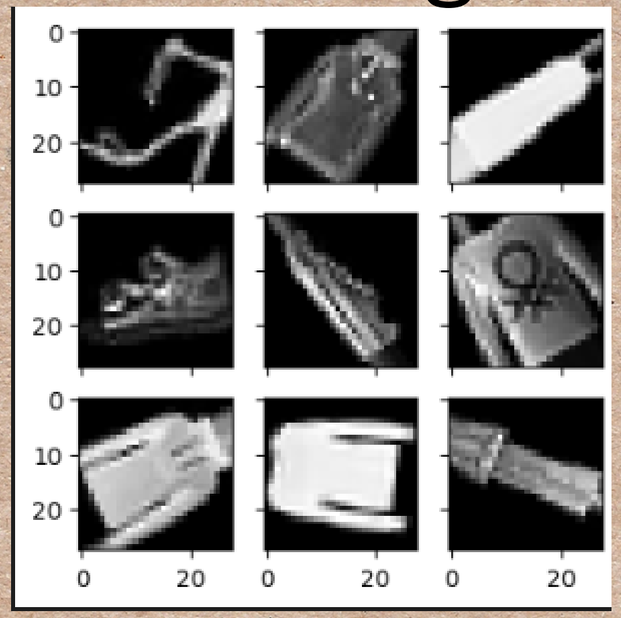
- WORKED BETTER IN A DEEPER CNN
- IMRPOVED DRASTICALLY TO 92%
- WILL BE ADDING THIS LAYER TO FINAL MODEL
- LOSS AND ACCURACY GRAPH SHOWS A DECENT FIT AND CONVERGING

- NN faces Covariate shift problem ; a change in the distribution of the input variables and also hidden inputs in the layers(Prof Andrew,Ng 2017)
- if the distribution keeps changing then them model will forever be trying to chase after a moving target of finding the best weights
- standardizes the inputs to a layer for each mini-batch making the distribution of the inputs to each layer more stable keeping it in a range so and generally so much lessens update
- arguments about this theory but scholars still agree on how the loss function is smoothened out and the model is able to converge faster generally.

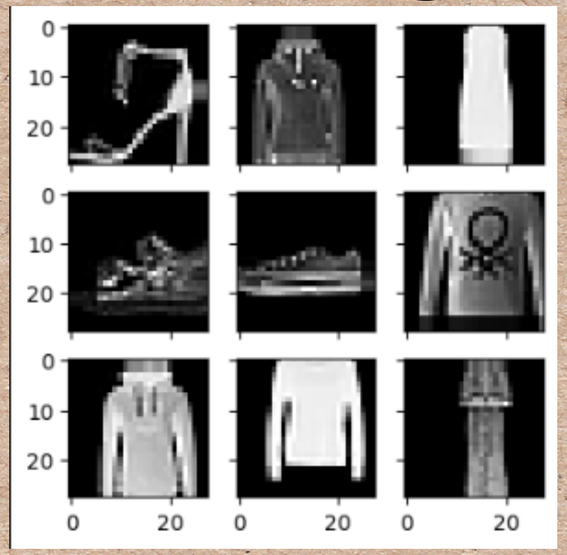
DATA AUGMENTATION (CNN)

ISSUE : DATA SCARCITY

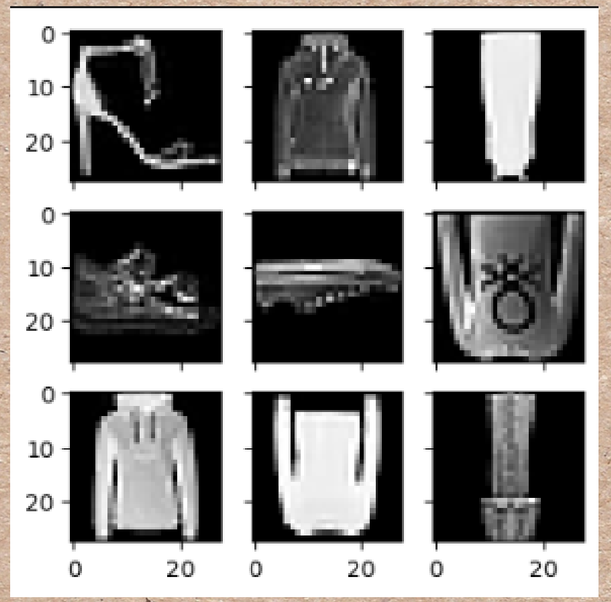
rotating



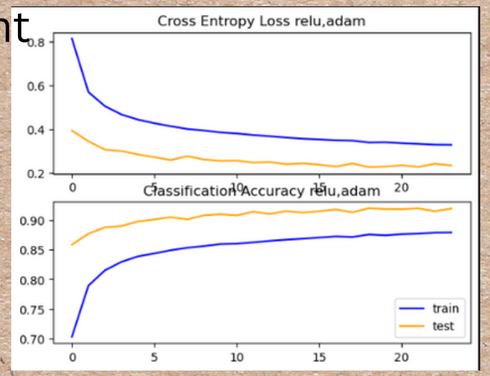
shifting



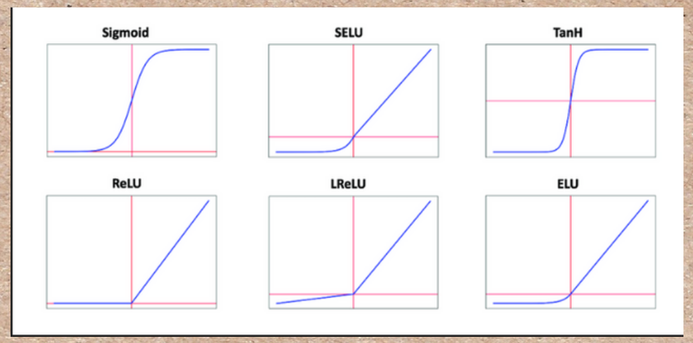
flipping



- Used ImageDataGenerator by keras which offered numerous features to augment
- Done rotating ,shifting and flipping separately on each copy of X_train
- did not zoom pictures or shear may not be able to see or cause distortion
- Tried Models w augmented images test acc increased
- total data points after adding all: (144000, 28, 28, 1) (10000, 28, 28, 1)
- Tried models on augmented images and evaluation test score increased
- validation set started getting higher scores than train



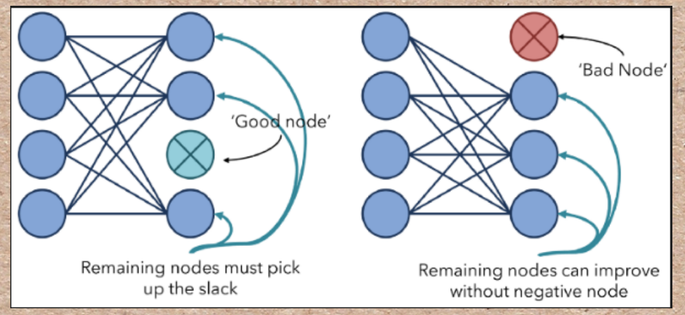
Explored various activation



		test acc	val loss
15	('elu', 'sgd')	89.690000	28.244346
16	('elu', 'rmsprop')	89.420003	30.105826
17	('elu', 'adagrad')	81.050003	52.059352
18	('elu', 'adadelta')	77.410001	59.613335
19	('elu', 'adamax')	89.730000	27.584884
20	('elu', 'nadam')	90.579998	25.781256
21	('tanh', 'adam')	89.459997	29.669052
22	('tanh', 'sgd')	88.730001	32.200432
23	('tanh', 'rmsprop')	88.300002	33.553246
24	('tanh', 'adagrad')	82.810003	47.884193
25	('tanh', 'adadelta')	76.690000	64.385569
26	('tanh', 'adamax')	90.899998	25.594884
27	('tanh', 'nadam')	89.330000	29.901490
28	('LeakyReLU', 'adam')	88.929999	29.089800
29	('LeakyReLU', 'sgd')	87.190002	35.391638
30	('LeakyReLU', 'rmsprop')	91.210002	24.427515
31	('LeakyReLU', 'adagrad')	84.340000	41.377118
32	('LeakyReLU', 'adadelta')	79.049999	58.410043
33	('LeakyReLU', 'adamax')	91.839999	22.693808
34	('LeakyReLU', 'nadam')	91.790003	22.800249

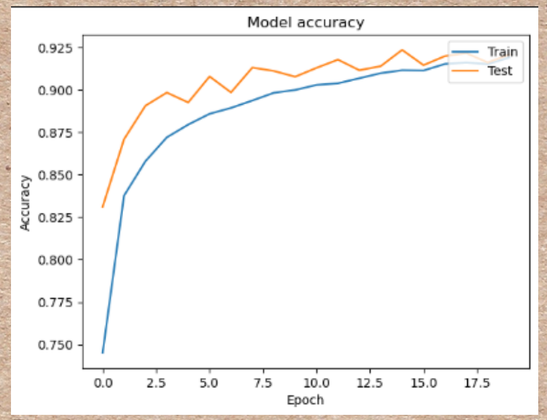
- Been comparing hypothesis like max pooling vs avg pooling / batch_size using linear activations
- Linear definitely will not be the best
- there are non-linear relations that can be drawn
- this requires a non-linear activation for such features to be captured
- Tanh did not do as well as compared to relu family perhaps due to vanishing gradient issue
- Sigmoid produces probability of 0 to 1 for last layer

DROPOUT LAYER



9	CNN relu adam w dropout	0.8826	1.016854	0.882927	0.8826	0.882460

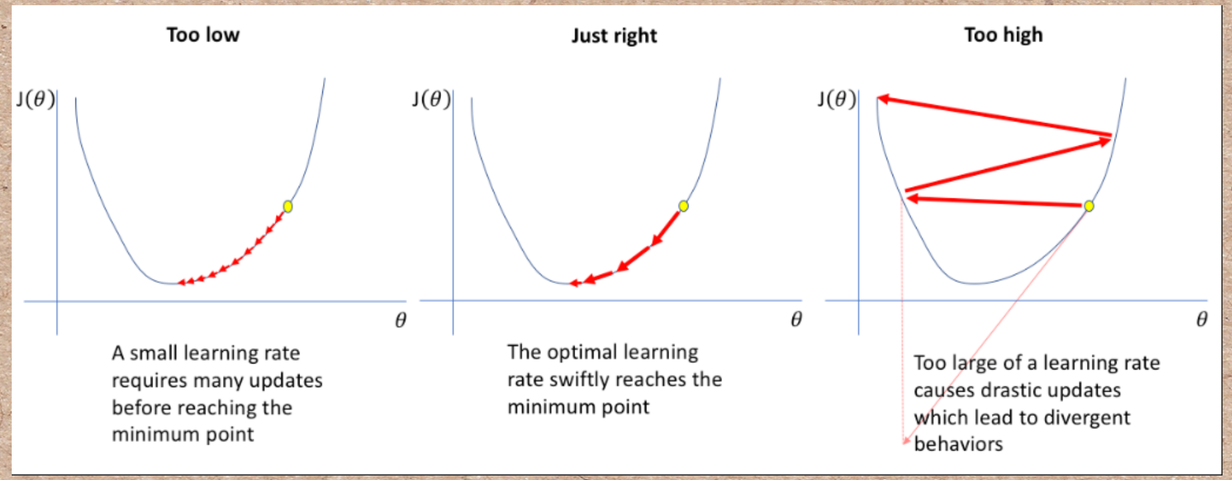
- Used relu adam for this hypothesis test
- Accuracy was roughly the same with and without
- practically has regularising effect
- even if good nodes is blocked it can try to learn the other nodes well
- 88% test accuracy was not really good



However it works well with batch normalization especially for deeper neural networks where it achieved 92.5% val acc

Explored various activation and optimize pair

12	('selu', 'adamax')	91.790003	22.987714
13	('selu', 'nadam')	90.490001	25.841162
14	('elu', 'adam')	89.940000	27.350724
15	('elu', 'sgd')	88.940001	30.256987
16	('elu', 'rmsprop')	90.030003	28.227046
17	('elu', 'adagrad')	81.750000	49.087885
18	('elu', 'adadelta')	78.009999	59.106594
19	('elu', 'adamax')	90.789998	24.654898
20	('elu', 'nadam')	90.619999	25.134510
21	('tanh', 'adam')	89.569998	28.989506
22	('tanh', 'sgd')	88.970000	30.908495
23	('tanh', 'rmsprop')	88.120002	33.597705
24	('tanh', 'adagrad')	81.989998	49.509081
25	('tanh', 'adadelta')	74.460000	69.739568
26	('tanh', 'adamax')	90.170002	26.991671
27	('tanh', 'nadam')	88.779998	31.099814
28	('LeakyReLU', 'adam')	91.070002	24.576949
32	('LeakyReLU', 'adadelta')	79.049999	58.410043
33	('LeakyReLU', 'adamax')	91.839999	22.693808
34	('LeakyReLU', 'nadam')	91.790003	22.800249



- Coded itertools function to tune best activation and optimizer pair
- highlighted means best performing pair
- weight initializations matter also affects accuracy so different run gets different results

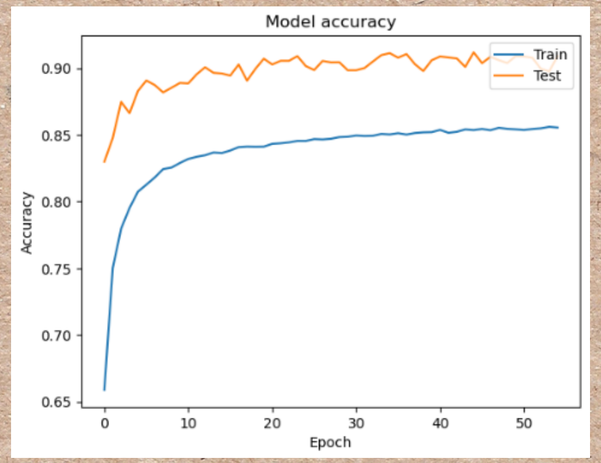
adam(adaptive movement): mix of Adagrad(updates learning rate) and RMSprop for scaling/normalizing gradient

adamax: very similar to adam but considers the infinite norm/ max of past gradient

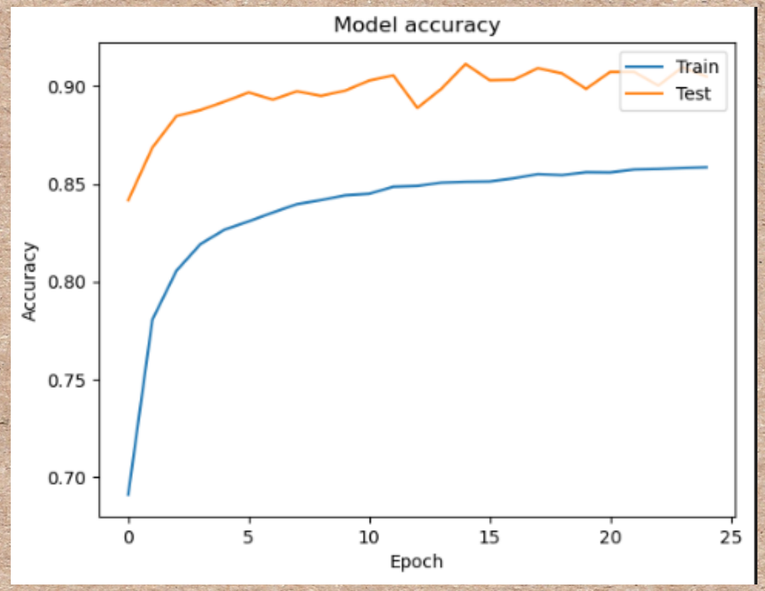
- CODED ITERTOOLS FUNCTION TO TUNE BEST ACTIVATION AND OPTIMIZER PAIR
- GENERALLY LINEAR UNIT (RELU.LEAKY RELU.SELU SOMETIEMS) FAMILY WINS WITH ADAM OR SIMILAR OPTIMIZERS WITH SIMILAR CONCEPTS SUCH AS ADAMAX
- COMPUTATIONALLY EXPENSIVE AND TIME CONSUMING
- INTRODUCED REDUCE LEARNING RATE ON PLATEAU CALLBACK AND EARLY STOPPING FOR THIS EXPERIMENT
- REDUCE LR HELPS REACH MINIMUM EFFICIENTLY AND SOMETIMES LOWER MINIMUM

REGULARIZATION

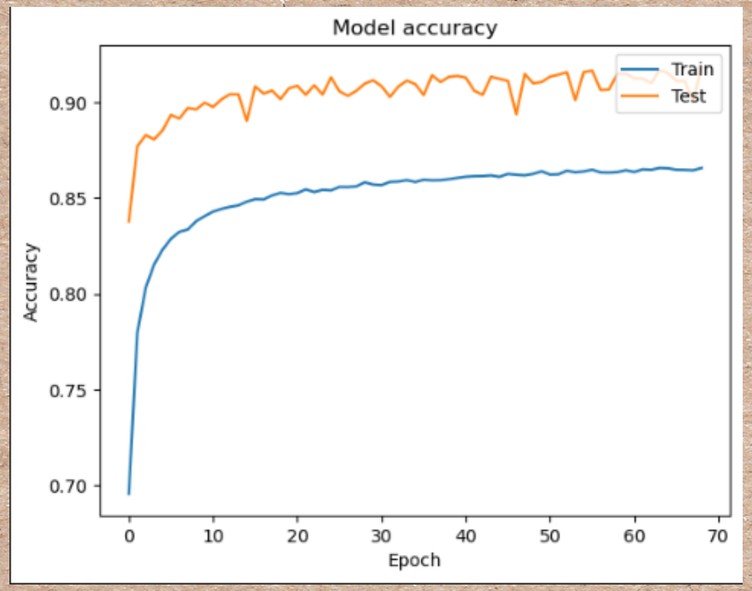
l1 90.90%



L2 90.05%



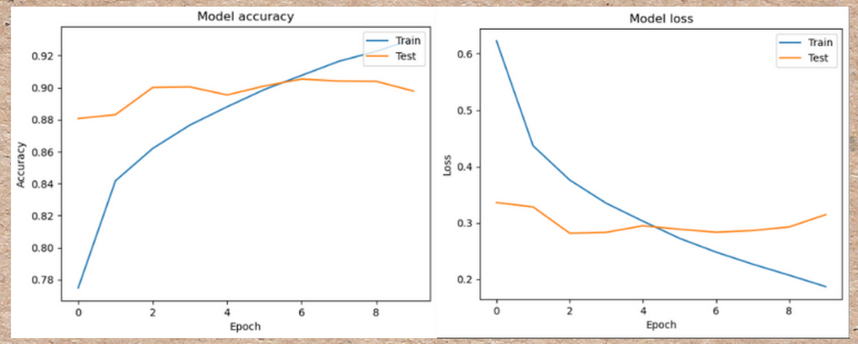
l1l2 91.86%



- Regularizing helps generalize better and not overfit on noisy data
- Felt that model was fine but still decided to try
- l1l2 got the highest accuracy but also more epochs
- went with l1l2 in the end

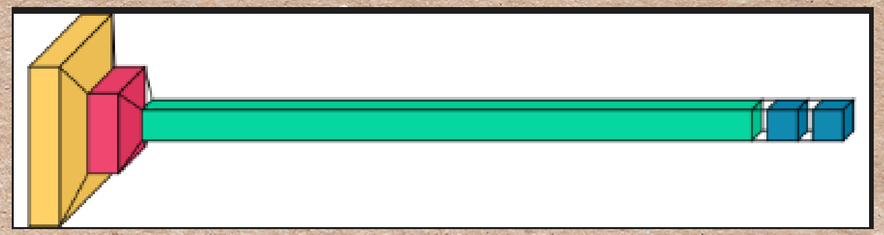
CHALLENGING MACHINE LEARNING MASTERY

Mastery's model

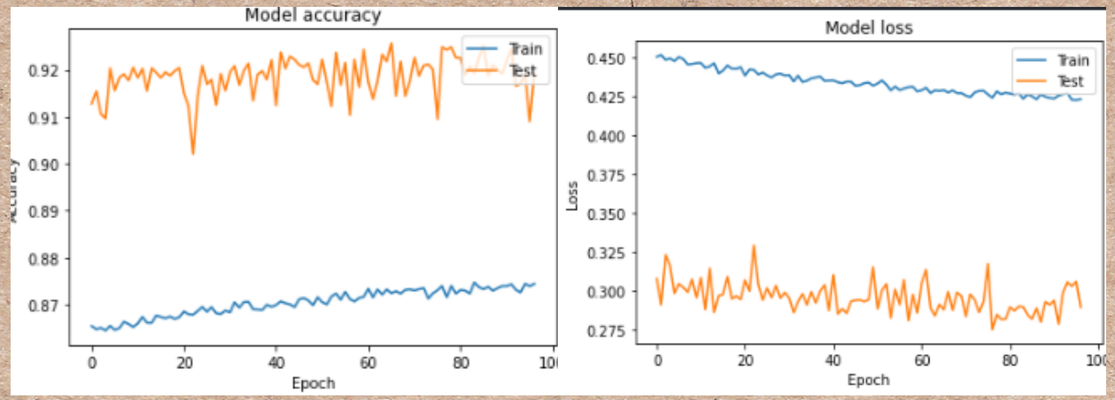


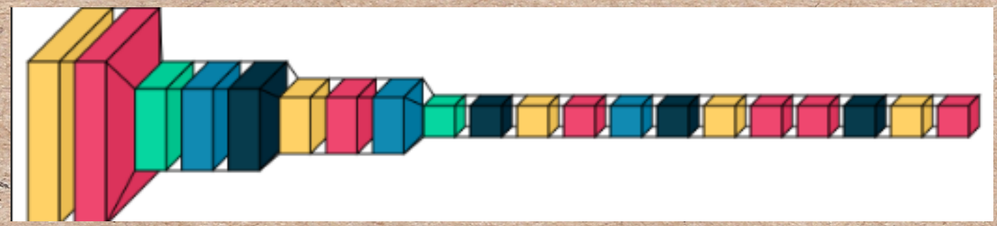

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320

max_pooling2d (MaxPooling2D  (None, 13, 13, 32)        0
)

flatten (Flatten)            (None, 5408)              0

dense (Dense)                (None, 100)               540900

dense_1 (Dense)              (None, 10)                1010

=================================================================
Total params: 542,230
Trainable params: 542,230
Non-trainable params: 0
```

- test accuracy of 0.872
- used SGD optimizer
- 542 290 params



My Final Model





- test accuracy of 0.9205
- leaky relu + adam
- decreasing filters with each layer
- 144 000 params

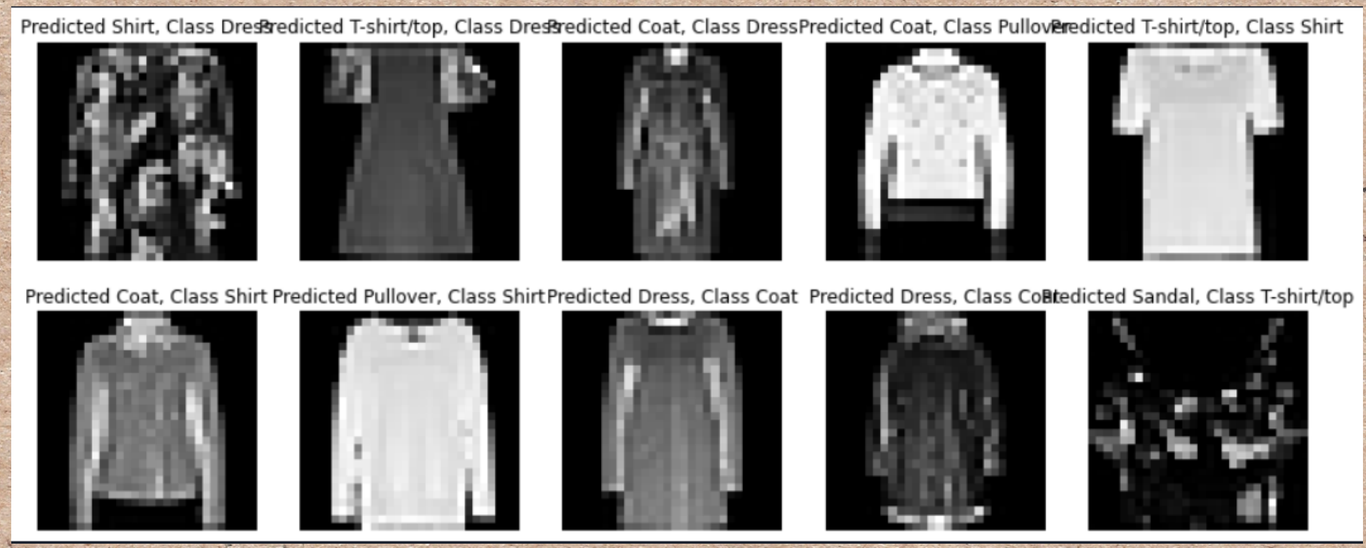
CONCLUSION

	Model	Accuracy	Loss	Precision	Recall	F1 Score
0	Dense 1 layer NN	0.8794	0.366588	0.883387	0.8794	0.879616
1	Dense NN 2 layer	0.8748	0.347432	0.874826	0.8748	0.873875
2	Dense NN 3 layer	0.8755	0.346159	0.875871	0.8755	0.874770
3	CNN linear activation	0.8195	0.499430	0.829953	0.8195	0.821822
4	CNN linear max pool	0.8778	0.700755	0.879536	0.8778	0.878137
5	CNN linear avg pool	0.8325	0.494453	0.831806	0.8325	0.829783
6	CNN linear w batch_norm	0.8210	0.516410	0.823537	0.8210	0.818529
7	linear batchnorm high epochs w rotated data aug	0.8301	0.491786	0.829276	0.8301	0.828320
8	CNN relu adam w dropout	0.8721	1.109318	0.872658	0.8721	0.872179
9	Machine Learning Mastery model	0.8721	1.109318	0.897904	0.8943	0.894330

	Model	Accuracy	Loss	Precision	Recall	F1 Score
0	Dense 1 layer NN	0.8811	0.362428	0.882335	0.8811	0.879924
1	linear batchnorm high epochs w rotated data aug	0.8351	0.476373	0.833472	0.8351	0.832964
2	Machine Learning Mastery model	0.8993	0.334355	0.901986	0.8993	0.900061
3	Fan in final model	0.9105	0.309365	0.911533	0.9105	0.910910
4	Fan out final model	0.9165	0.307018	0.916260	0.9165	0.915718

- kept track of improvements what went well
- fanning out model was decent
- higher params does not necessarily improve performance
- leaky relu with adam/adamax worked best for me
- tested most of CNN's topics including data augmentation, pooling, batch normalization, dropout, and regularization
-

Wrongly predicted images



- Reasonable for model to misclassify
- Difficult even for the naked eye to distinguish
- Dress,Coat and shirt looks really similar to me
- Overall I think the model did a decent job classifying fashion mnist