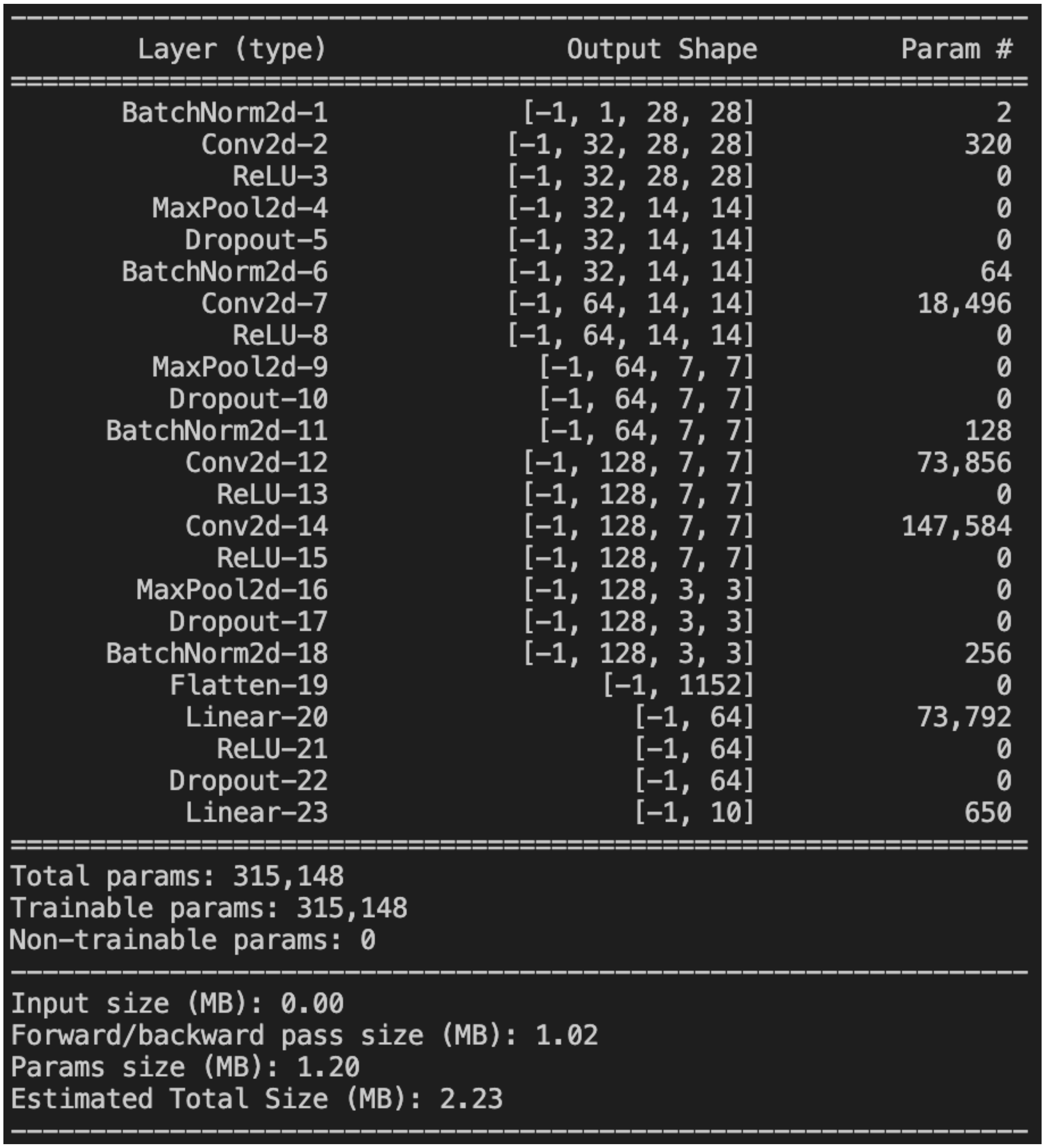
Model Overview



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
       BatchNorm2d-1            [-1, 1, 28, 28]               2
            Conv2d-2           [-1, 32, 28, 28]             320
              ReLU-3           [-1, 32, 28, 28]               0
         MaxPool2d-4           [-1, 32, 14, 14]               0
           Dropout-5           [-1, 32, 14, 14]               0
       BatchNorm2d-6           [-1, 32, 14, 14]              64
            Conv2d-7           [-1, 64, 14, 14]          18,496
              ReLU-8           [-1, 64, 14, 14]               0
         MaxPool2d-9             [-1, 64, 7, 7]               0
          Dropout-10             [-1, 64, 7, 7]               0
      BatchNorm2d-11             [-1, 64, 7, 7]             128
           Conv2d-12            [-1, 128, 7, 7]          73,856
             ReLU-13            [-1, 128, 7, 7]               0
           Conv2d-14            [-1, 128, 7, 7]         147,584
             ReLU-15            [-1, 128, 7, 7]               0
        MaxPool2d-16            [-1, 128, 3, 3]               0
          Dropout-17            [-1, 128, 3, 3]               0
      BatchNorm2d-18            [-1, 128, 3, 3]             256
          Flatten-19                 [-1, 1152]               0
           Linear-20                   [-1, 64]          73,792
             ReLU-21                   [-1, 64]               0
          Dropout-22                   [-1, 64]               0
           Linear-23                   [-1, 10]             650
================================================================
Total params: 315,148
Trainable params: 315,148
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 1.02
Params size (MB): 1.20
Estimated Total Size (MB): 2.23
----------------------------------------------------------------
```

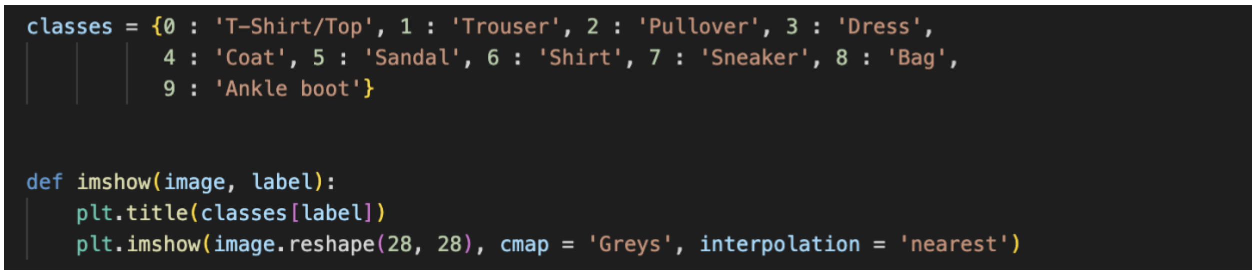
The convolutional layers of my model take a batch normalization operation to normal input image values followed by a convolutional operation (which is learned by the model) a ReLU activation function, providing non-linearity to the model a max pool operation followed by a dropout layer. The model is based on “Fashion MNIST – image classification w/ CNN” (Barbosa J. P. 2022) with further enhancement via grid search, for hyperparameters. (Radhakrishnan P. 2017)

Data

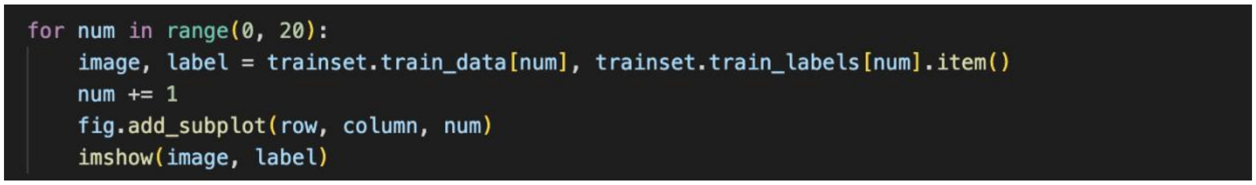
Dataset:

Fashion MNIST Dataset is taken from Zalando’s article images. 60,000 of the images are training examples; 10,000 are for testing. Testing breaks down into validation and testing. Each image is a 28 x 28 greyscale image falling into 10 possible classifications.

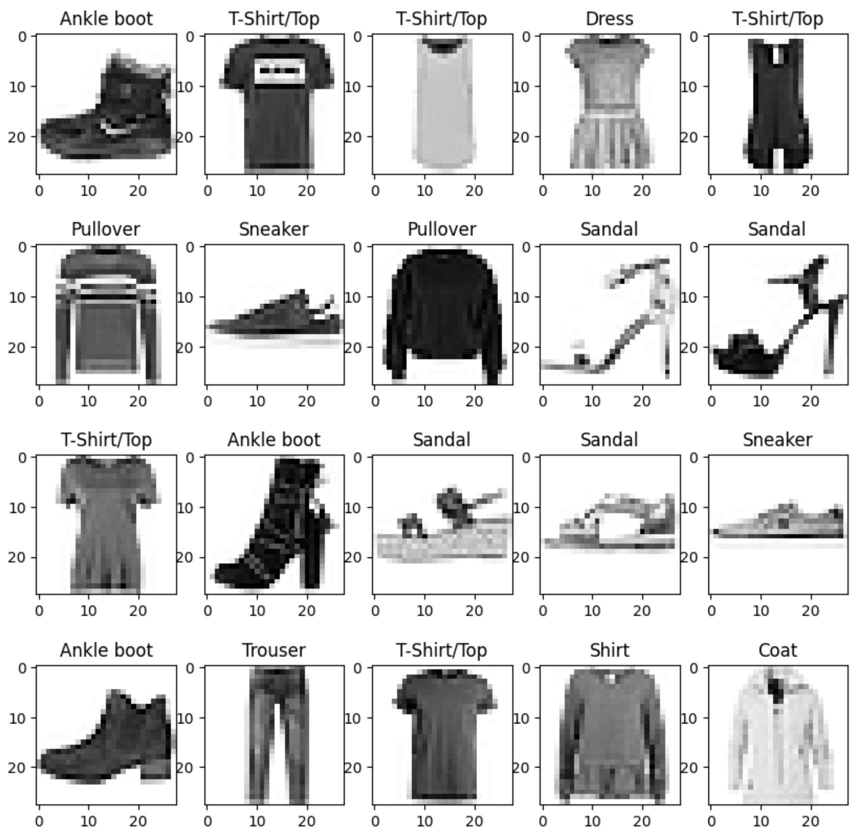
Data Visualization:



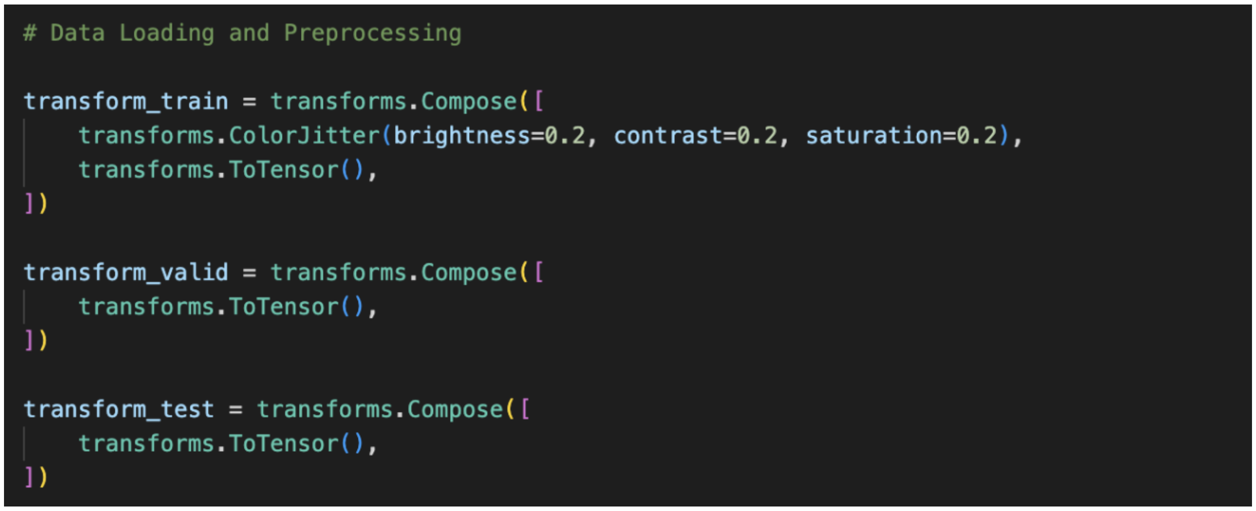
```
classes = {0 : 'T-Shirt/Top', 1 : 'Trouser', 2 : 'Pullover', 3 : 'Dress',
           4 : 'Coat', 5 : 'Sandal', 6 : 'Shirt', 7 : 'Sneaker', 8 : 'Bag',
           9 : 'Ankle boot'}


def imshow(image, label):
    plt.title(classes[label])
    plt.imshow(image.reshape(28, 28), cmap = 'Greys', interpolation = 'nearest')
```



```
for num in range(0, 20):
    image, label = trainset.train_data[num], trainset.train_labels[num].item()
    num += 1
    fig.add_subplot(row, column, num)
    imshow(image, label)
```



Data Augmentation:



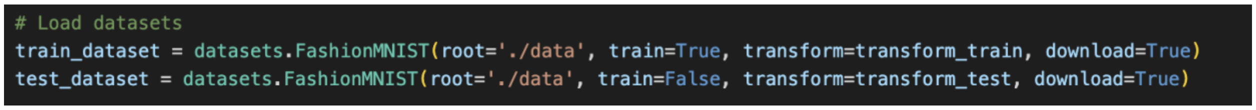
```
# Data Loading and Preprocessing

transform_train = transforms.Compose([
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2),
    transforms.ToTensor(),
])

transform_valid = transforms.Compose([
    transforms.ToTensor(),
])

transform_test = transforms.Compose([
    transforms.ToTensor(),
])
```

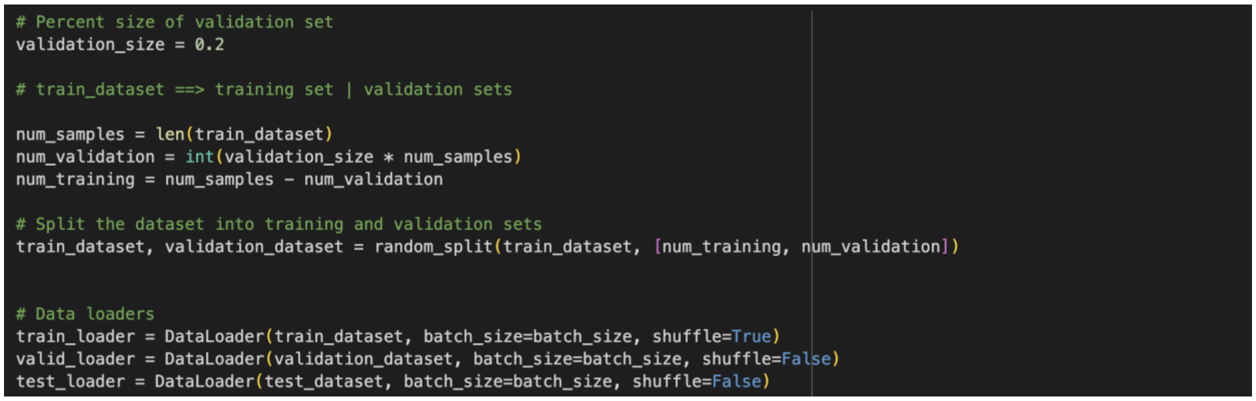
Color Jitter –

- **Brightness=0.2**: Adjusting the brightness of the image ± 0.2.
- **Contrast=0.2**: Controls the range for adjusting the contrast of the image again ± 0.2
- **Saturation=0.2**: Adjusts the saturation of the image. (The intensity of colors in the image).

To Tensor –

A transformation used when working with image data as it takes in a PIL image or NumPy array and converts the image into PyTorch tensor. It also takes the pixel from a range of [0,255] to the range [0.0,1.0]. This normalization process is essential as it helps ensure the input values are within a similar scale and thus improve training stability and convergence. Lastly the operation reorders the dimensions of the image to be compatible with what the PyTorch framework expects. (The PyTorch Contributors n.d.,) Training & Testing Dataset


```
# Load datasets
train_dataset = datasets.FashionMNIST(root='./data', train=True, transform=transform_train, download=True)
test_dataset = datasets.FashionMNIST(root='./data', train=False, transform=transform_test, download=True)
```


```
# Percent size of validation set
validation_size = 0.2

# train_dataset ==> training set | validation sets

num_samples = len(train_dataset)
num_validation = int(validation_size * num_samples)
num_training = num_samples - num_validation

# Split the dataset into training and validation sets
train_dataset, validation_dataset = random_split(train_dataset, [num_training, num_validation])


# Data loaders
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
valid_loader = DataLoader(validation_dataset, batch_size=batch_size, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
```

I breakdown the dataset into 3 distinct subsets:

1. The Training Set:

This is the subset of the data used to train the model. The model learns from the patterns, relationships, and features present in the train_dataset. The model adjusts its parameters to minimize the error or loss function on the train_dataset.

2. Validation Set:

This set is used to tune hyperparameters and assess the model's performance during training. While training the model on the train_dataset, I evaluate its performance on the validation_dataset which is a 20/80 split from the original training set (20% for validation; 80% for training). Why do this? This helps avoid overfitting so that hopefully the model is better able to generalize.

3. Test Set:

This set is used to provide an unbiased evaluation of a final model performance on the training and validation data. Once the model is trained the test_dataset results give an estimate of how well the model is likely to perform on new, unseen data.

Model Architecture Explained

Batch Normalization:

Batch normalization was introduced by Sergey Ioffe and Christian Szegedy in their 2015 paper titled "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift." (Ioffe S., Szegedy C, 2015) The benefits of Batch Normalization are: that it improves the training stability by mitigating the problem of vanishing/exploding gradients by ensuring that the inputs are normalized. and speed; it reduces the internal covariate shift keeping the distribution of each layer consistent throughout training; and offers better generalization on unseen data.

The idea behind batch normalization is to normalize the inputs of each layer in a neural network by subtracting the mean and dividing by the standard deviation of the mini-batch during training. This is done for each feature independently, so the normalization is applied along each feature dimension.

Mathematically,

$$BN(x_i) = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

Where:

x_i is the input to the layer for a particular example i_i,

μ is the mean of the mini-batch,

σ is the standard deviation of the mini-batch,

ϵ is a small constant added for numerical stability.

Convolutional Layer:

Convolutional layers are designed to leverage the spatial relationships and local patterns in the input data, making them highly effective for tasks such as image classification and object detection. They allow the network to automatically learn and extract hierarchical features from the input images.

A convolutional filter, kernel or filter, is a small matrix used to extract features from input data, images. The convolutional filters allow for convolutional layers of a neural network to detect specific patterns or features within an image.

To begin with a convolutional operation is applied to the input image. The convolutional filter slides over the image at each position calculating the dot product between the filter and region currently being covered. The filter is a learned parameter, meaning that the model, through a process of backpropagation finds the filter that extracts the most important feature and feeds it into a simple multi-layered perceptron.

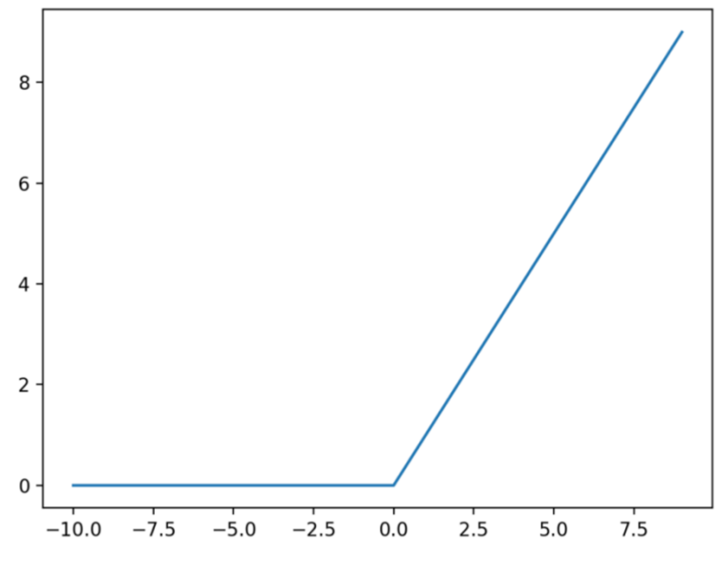
Other important operation within my Model / CNN in general are the strides, padding and pooling. A stride is the step size which a convolutional kernel moves across the image, therefore meaning large strides smaller feature mappings. Adding padding around a givens input adds pixels around the image which allows convolution operation is able to be applied to border pixels without losing spatial information. Finally pooling (max pooling in my case) reduces the spatial resolution of the input while maintaining the important information.

Activation Function:

Activation functions are used to add non-linearity to network allowing the network to learn complex patterns and representations.

ReLU (Rectified Linear Unit) is the activation function I used to add non-linearity to my model. The function takes an input x and if x is less than 0 then x is automatically set to 0 and if x is positive then x is kept the same.

$$f(x) = \max(0, x)$$



(Paperswithcode, n.d.,)

Why I choose ReLU:

- Non-Linearity – enables network to learn from and adapt to complex relationships in data

- Computational Efficiency – simple thresholding function hence easy and faster to train compared to other activation functions.
- Sparse Activation – negative values set to zero which leads to them not contributing finally output. Allowing network to learn more complex representations with fewer parameters.

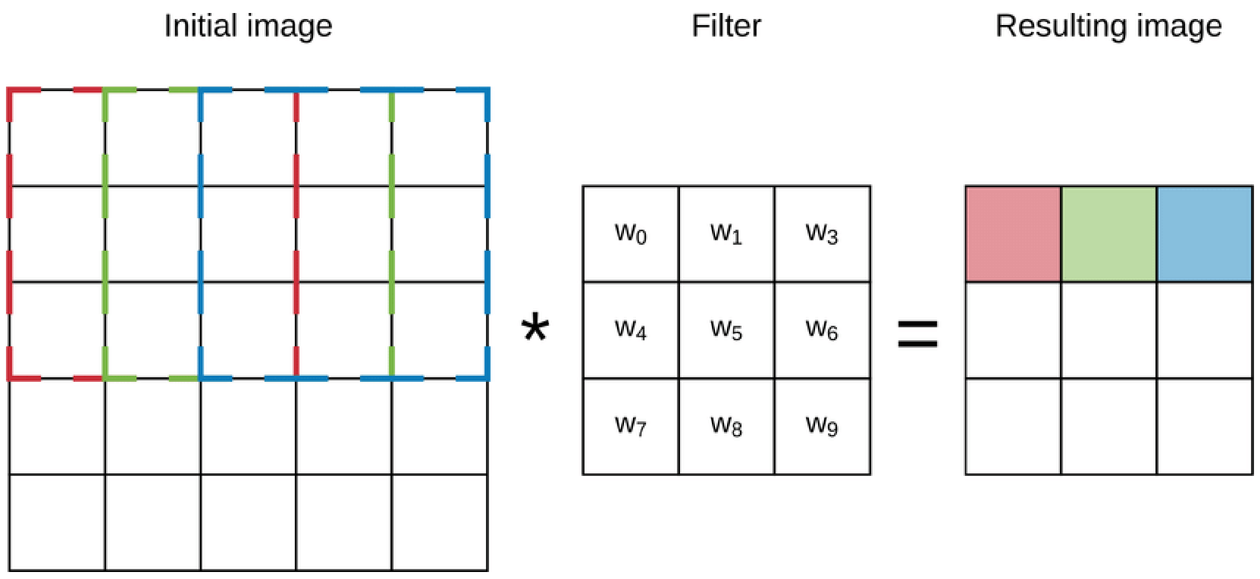
Issues with ReLU:

- Dead neurons – some neurons always output zero (dead neurons). If a large gradient flows through a ReLU neuron during backpropagation, it can update the weights in a way that the neuron will always output zero.
- Unbound Activation - ReLU doesn't have an upper bound meaning neuron values can become very large. However, I mitigate this with the batch normalization layers.

Max Pooling:

Max pooling is a down sampling operation meaning it reduces the spatial dimension of the input feature map. Max pooling takes the maximum value form a group of neighboring pixels the rest are not taken into account.

Within a pooling window (small rectangular pooling window e.g., 3x3) the max values at each position are taken and the sliding window, slides pooling window over input with specified stride, (not overlapping) this is repeated until the whole input is sampled.



(Kadam R. 2021)

$$Max\ Pooling\ (x)_{i,j} = \max\left(x_{2i,2j}, x_{2i,2j+1}, x_{2i+1,2j}, x_{2i+1,2j+1}\right)$$

Where x is that the input feature.

Why use Max Pooling?

- Translation invariance – max pool enables the model to become invariant to small changes in inputs; making model more robust
- Reduction of spatial dimensions – reduces the total number of parameters making model more computationally efficient
- Feature selection – max values hence model maintains most prominent features of each region

However, this is not perfect as some information is lost as the non-maximum values are discarded.

Dropout:

Dropout is a regularization technique used to prevent overfitting and improve the generalization of the model. It was introduced by Geoffrey Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov in their 2012 paper titled "Improving neural networks by preventing co-adaptation of feature detectors." (Hinton G., Srivastava N., Krizhevsky A., Sutskever I., Salakhutdinov R., 2012)

Dropout randomly "dropouts" a subset of neuron independent for each example in a minibatch for each forward and backwards pass through the network.

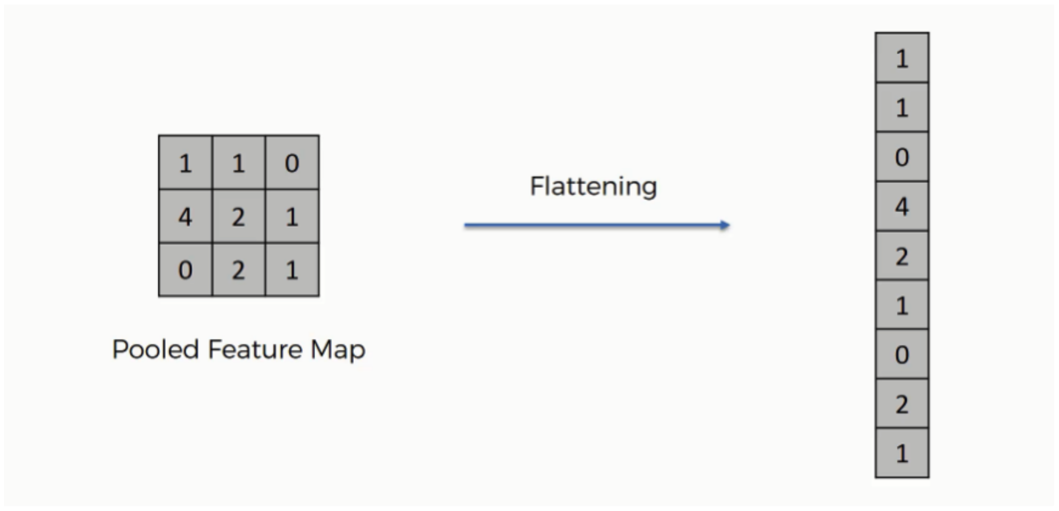
As during training some neurons are not active when we run inference with have to account for the fact that more neurons are active so the outputs have to be scaled by the dropout probability.

Why use Dropout?

- Regularization – prevents network from relying too much of specific neuron and promote a more robust model that is able to generalize
- Ensemble learning – dropping out various subset of neurons during training which can be thought of as training various models with shared parameters. This can be thought of as an ensemble learning approach.
- Reduced overfitting – makes the model less sensitive to noise and reduces the risk of overfitting.

Flatten

This operation takes multi-dimensional input data and converts it into a one-dimensional array, essentially "flattening it". The subsequent output is then able to be feed into a fully connected layer for more processing and the finally classification.



(SuperDataScience Team 2018)

Linear

A linear layer aka fully connected layer, connects each neuron is connected to every other neuron.

A connected layer can be mathematically represented as,

$$Output = Activation(\sum_i Input_i * Weight_i + Bias)$$

Where:

- Input $_i$ is the input from the *i*-th neuron in the previous layer,
- Weight $_i$ is the weight associated with the connection between the *i*-th neuron in the previous layer and the current neuron,
- Bias is a bias term added to the weighted sum,
- Activation is an activation function applied element-wise to the result.

Calculating Output Sizes

Input size: W x W x D
D_{out}: Number of Kernels
F: Spatial Size
S: Stride

P: Padding

Convolutional Layer:

$$W_{out} = \frac{W - F + 2P}{S} + 1$$

Padding Layer:

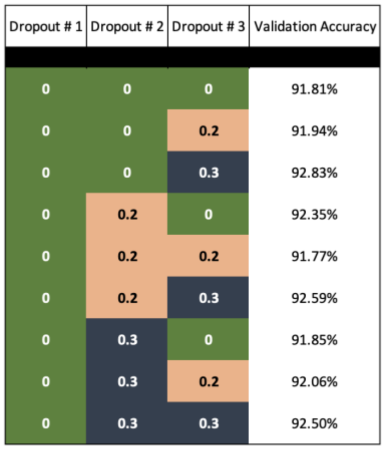
$$W_{out} = \frac{W - F}{S} + 1$$

(Mishra M. 2020)

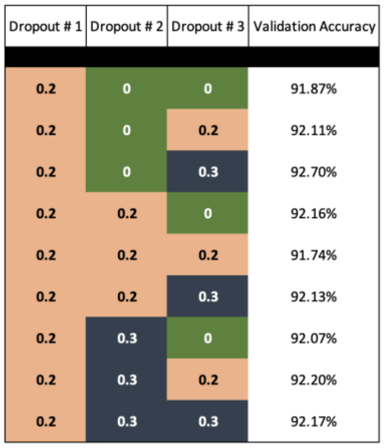
Grid Search and Hyperparameters

What is Grid Search?

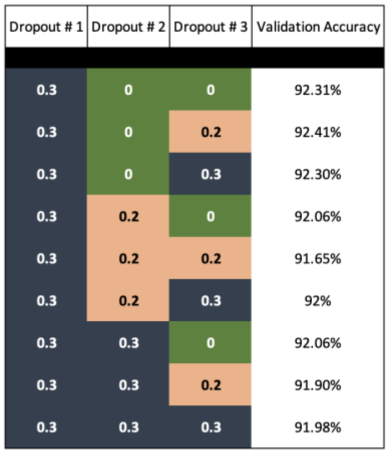
Grid search is a way of exhaustively testing every possible combination of hyperparameters values and then eventually selecting the best performance parameters. This is however more computationally expensive so logical could tests all possible values for each possible hyperparameters. There I choose to test varying 1. dropout thresholds, and 2. Final flatten layer size. (see results below). Each potential variant train on 10 epochs. The best hyperparameters are then train for a full 100 epochs.



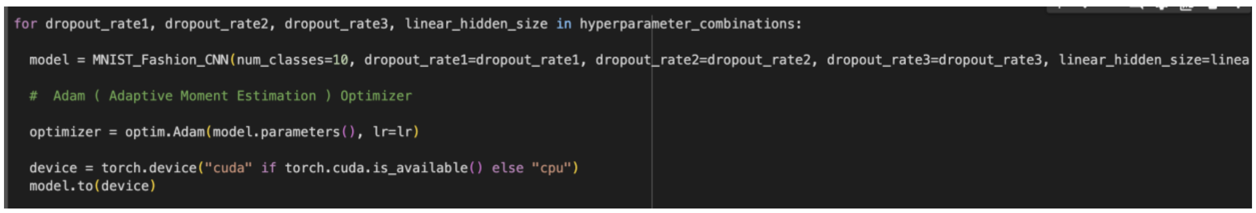
Dropout # 1	Dropout # 2	Dropout # 3	Validation Accuracy
0	0	0	91.81%
0	0	0.2	91.94%
0	0	0.3	92.83%
0	0.2	0	92.35%
0	0.2	0.2	91.77%
0	0.2	0.3	92.59%
0	0.3	0	91.85%
0	0.3	0.2	92.06%
0	0.3	0.3	92.50%



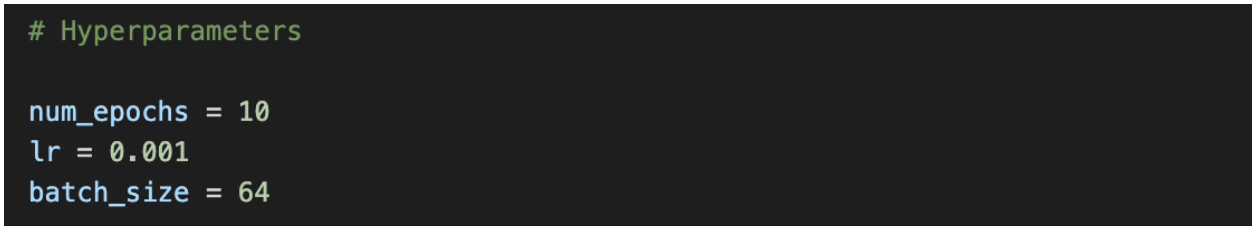
Dropout # 1	Dropout # 2	Dropout # 3	Validation Accuracy
0.2	0	0	91.87%
0.2	0	0.2	92.11%
0.2	0	0.3	92.70%
0.2	0.2	0	92.16%
0.2	0.2	0.2	91.74%
0.2	0.2	0.3	92.13%
0.2	0.3	0	92.07%
0.2	0.3	0.2	92.20%
0.2	0.3	0.3	92.17%



Dropout # 1	Dropout # 2	Dropout # 3	Validation Accuracy
0.3	0	0	92.31%
0.3	0	0.2	92.41%
0.3	0	0.3	92.30%
0.3	0.2	0	92.06%
0.3	0.2	0.2	91.65%
0.3	0.2	0.3	92%
0.3	0.3	0	92.06%
0.3	0.3	0.2	91.90%
0.3	0.3	0.3	91.98%



```
for dropout_rate1, dropout_rate2, dropout_rate3, linear_hidden_size in hyperparameter_combinations:

  model = MNIST_Fashion_CNN(num_classes=10, dropout_rate1=dropout_rate1, dropout_rate2=dropout_rate2, dropout_rate3=dropout_rate3, linear_hidden_size=linea

  #  Adam ( Adaptive Moment Estimation ) Optimizer

  optimizer = optim.Adam(model.parameters(), lr=lr)

  device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
  model.to(device)
```



```
# Hyperparameters

num_epochs = 10
lr = 0.001
batch_size = 64
```

Num_epochs – number of epochs, which is the number of times the whole training data is shown to the network during training.

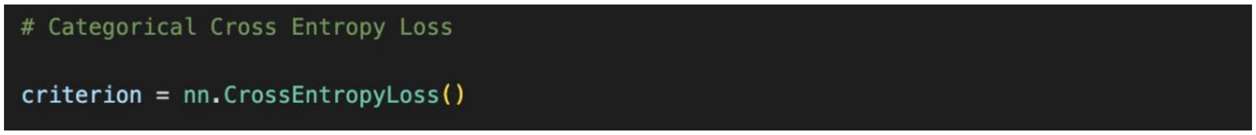
Lr – is the learning rate which influence how fast the network updates its parameters.

Batch_size – this is the number of sub-samples given to the network after which parameter update happens.

Loss Functions:

What is a loss function?

- A loss function is a way to mathematical quantifies the difference between predicted and actual values in a machine learning model. It measures the model's performance and guides the optimization process by providing feedback on how well the model fits the data.



```
# Categorical Cross Entropy Loss

criterion = nn.CrossEntropyLoss()
```

1. Categorical Cross Entropy Loss

used in multiclass classification and SoftMax regression. The categorical cross entropy function takes the negation of the summation of all instances, of a given class, multiplied by the log of the prediction given you a loss. (how accurate the model is for the given class)

$$Loss = -\sum_{j=1}^{k} y_j \log(\hat{y}_j)$$

Then to get a cumulative cost function that the network can optimize over you take the previous calculated loss functions, for each given class, and add them all together and divide by the total number of independent classes within the dataset.

$$Cost = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{k}[y_{ij}\ \log(\hat{y}_{ij})]$$

Where:

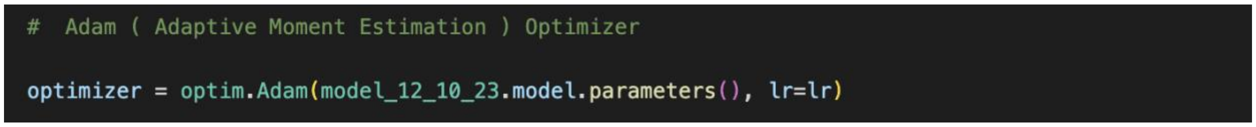
- K is classes,
- y = actual value
- y hat = Neural network prediction

in multi-class classification last neuron uses the SoftMax activation function.

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

(Shankar 2023)

Optimizer:



```
#  Adam ( Adaptive Moment Estimation ) Optimizer

optimizer = optim.Adam(model_12_10_23.model.parameters(), lr=lr)
```

Optimizer are used to change the weights and learning rate of a neural network in order to minimize the loss (the difference between the actual input and the predicted input). There several optimization algorithms each with its own disadvantages and advantages.

Gradient Descent is the simplest optimization algorithm which is easy to compute, it uses the first order derivative of the loss function and then uses backpropagation to compute the loss from on layer to the next. However, the algorithm may not reach the global minima instead believing a local minimum to be the optima. (Doshi S. 2019)

So, this is where the Adam Optimizer comes into play.

How does it work?

Adam (Adaptive Moment Estimation) does two things:

Works with momentums of first and second order. The basic idea behind Adam is that we don't want to roll so as to miss the minimum, we decrease the velocity as we descend the gradient.

Adams stores both exponentially decaying average of past square gradients and exponentially decaying average of past gradients M(t).

M(t) & V(t) take on the values of the first moment (Mean) and the second moment (uncentered variance) of the gradients.

$$\hat{m}_t = \frac{m_t}{1 - \beta_2^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

And to update parameters:

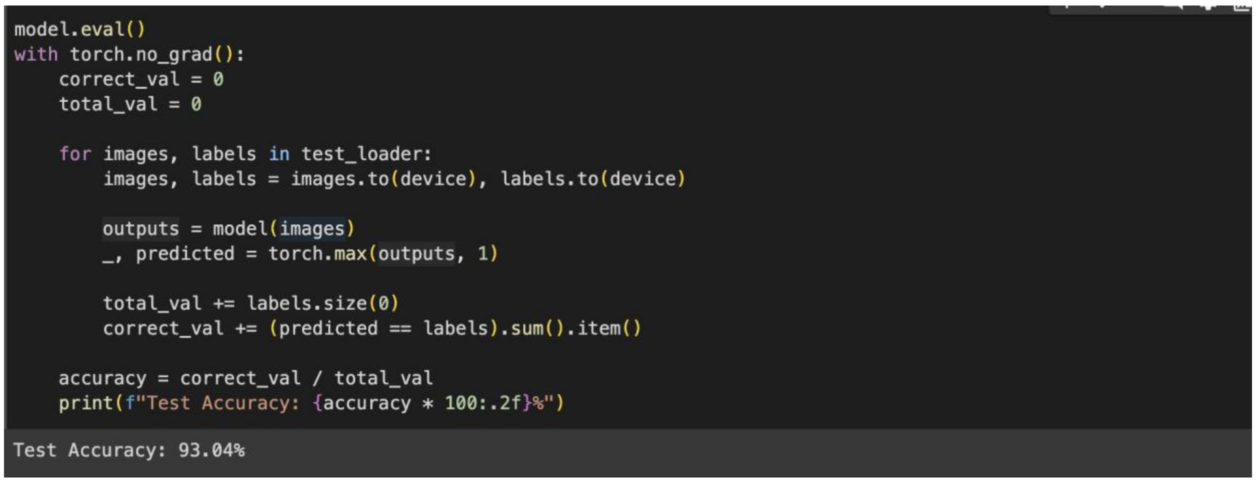
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

The values for beta are 0.9, 0.99999 for beta 2 and (10 x exp (-8)) for epsilon.

The biggest advantage of the Adam optimizer is its speed and the fact that it converges rapidly, however it is computationally costly. (Doshi S. 2019)

Conclusion & Analysis

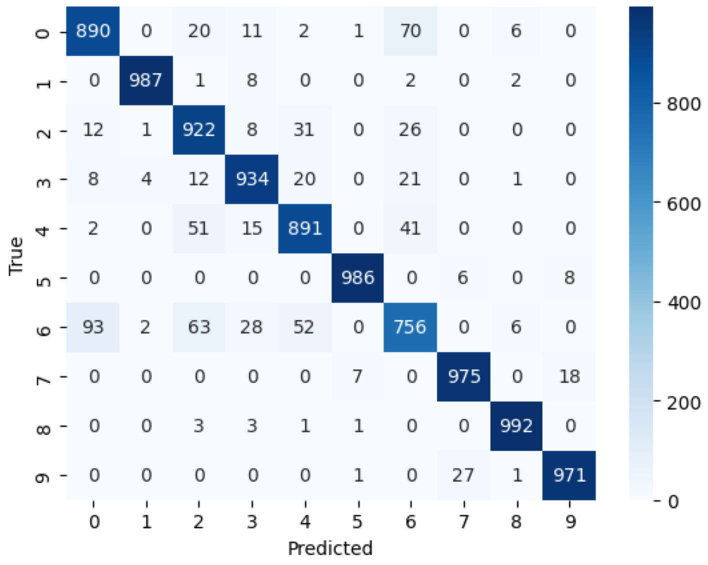
Testing Accuracy:



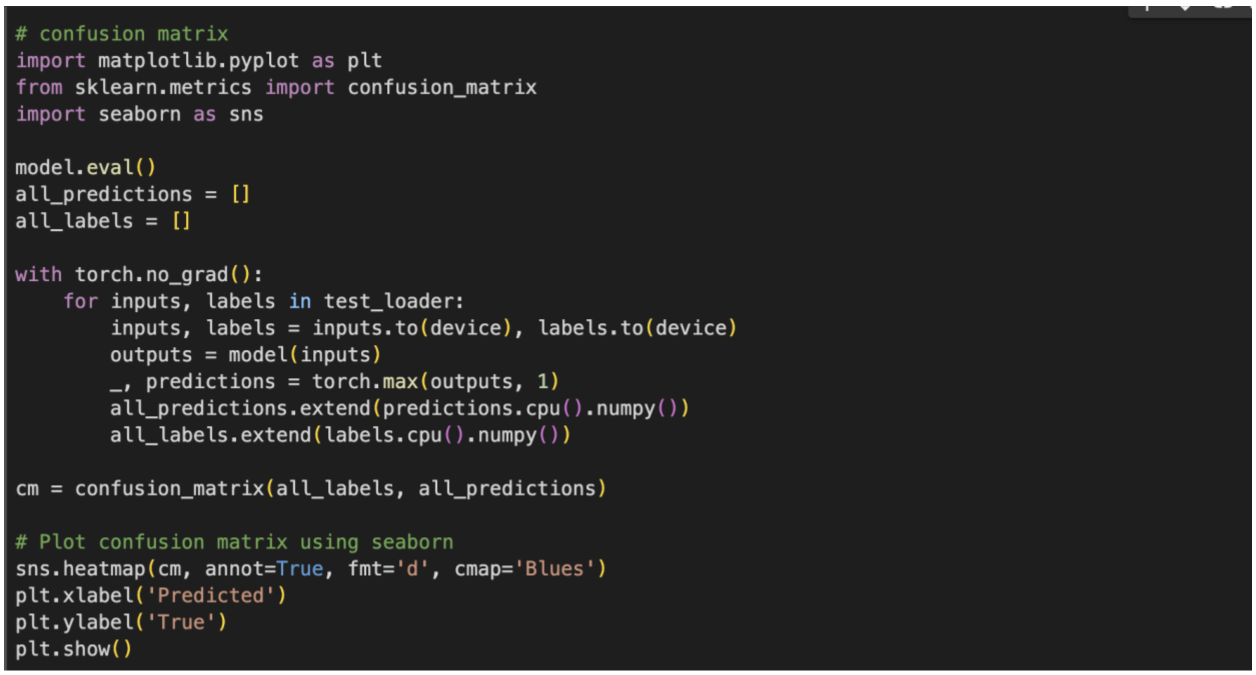
```
model.eval()
with torch.no_grad():
    correct_val = 0
    total_val = 0

    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)

        outputs = model(images)
        _, predicted = torch.max(outputs, 1)

        total_val += labels.size(0)
        correct_val += (predicted == labels).sum().item()

    accuracy = correct_val / total_val
    print(f"Test Accuracy: {accuracy * 100:.2f}%")
```

```
Test Accuracy: 93.04%
```

Confusion Matrix (N x N matrix used to evaluate performance of a classification model; the matrix compares actual target values vs predicted values).

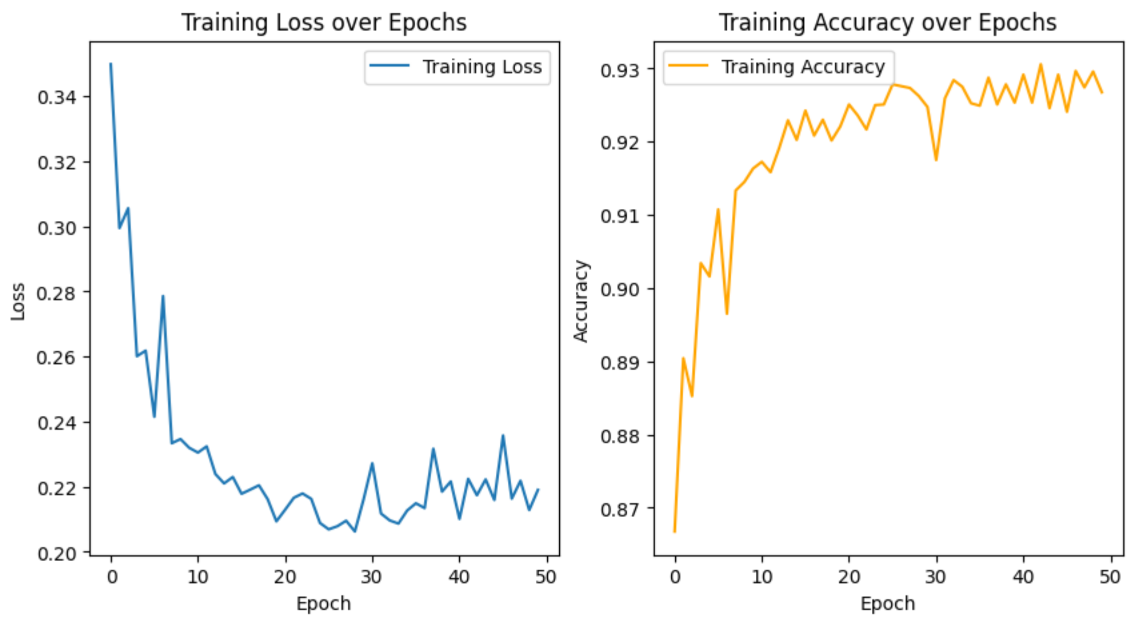


True \ Predicted	0	1	2	3	4	5	6	7	8	9
0	890	0	20	11	2	1	70	0	6	0
1	0	987	1	8	0	0	2	0	2	0
2	12	1	922	8	31	0	26	0	0	0
3	8	4	12	934	20	0	21	0	1	0
4	2	0	51	15	891	0	41	0	0	0
5	0	0	0	0	0	986	0	6	0	8
6	93	2	63	28	52	0	756	0	6	0
7	0	0	0	0	0	7	0	975	0	18
8	0	0	3	3	1	1	0	0	992	0
9	0	0	0	0	0	1	0	27	1	971

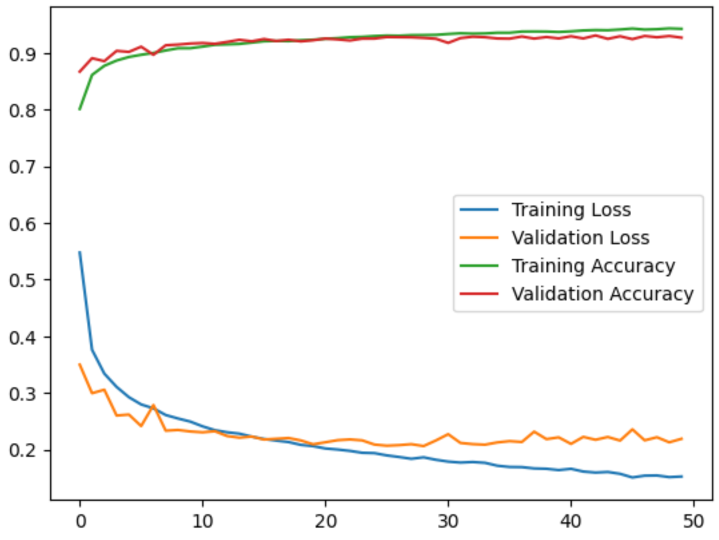


```python
# confusion matrix
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import seaborn as sns

model.eval()
all_predictions = []
all_labels = []

with torch.no_grad():
    for inputs, labels in test_loader:
        inputs, labels = inputs.to(device), labels.to(device)
        outputs = model(inputs)
        _, predictions = torch.max(outputs, 1)
        all_predictions.extend(predictions.cpu().numpy())
        all_labels.extend(labels.cpu().numpy())

cm = confusion_matrix(all_labels, all_predictions)

# Plot confusion matrix using seaborn
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

Loss and Accuracy Plots: Model loss decreases up until 20 ~ where it begins to stagnate while accuracy seems to top out to around 93 %



Learning curve: Training accuracy and validation accuracy remain relatively in line showing that the model is indeed generalizing and not just fitting to the data. However, validation loss at around 20 epochs does show slight signs of overfitting. Little to no benefit of training pass 20 epochs.



Activation maps

Block_1: low level features extraction

Activation 1
Activation 10
Activation 19
Activation 28

References:

Shankar 2023, *Understanding Loss Function in Deep Learning*, viewed 1st Dec 2023, < https://www.analyticsvidhya.com/blog/2022/06/understanding-loss-function-in-deep-learning/ >

Doshi S. 2019, *Various Optimization Algorithms For Training Neural Network*, viewed 2nd Dec 2023, < https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6 >

Mishra M. 2020, *Convolutional Neural Networks*, *Explained*, viewed 5th Dec 2023, < https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939 >

The PyTorch Contributors n.d., *Transforming and Augmenting Image*, viewed 5th Dec 2023, < https://pytorch.org/vision/stable/transforms.html >

Barbosa J. P. 2022, *Fashion MNIST - Image Classification w/CNNs (~94%)*, viewed 6th Dec 2023, < https://www.kaggle.com/code/jonaspalucibarbosa/fashion-mnist-image-classification-w-cnns-94 >

Radhakrishnan P. 2017, *What are Hyperparameters? and How to tune the Hyperparameters in a Deep Neural Network?*, viewed 6th Dec 2023, < https://towardsdatascience.com/what-are-hyperparameters-and-how-to-tune-the-hyperparameters-in-a-deep-neural-network-d0604917584a#:~:text=Hyperparameters%20are%20the%20variables%20which,optimizing%20the%20weights%20and%20bias >

Paperswithcode, n.d., *Rectified Linear Units*, Viewed 7th Dec 2023, < https://production-media.paperswithcode.com/methods/Screen_Shot_2020-05-27_at_1.47.40_PM.png >

Kadam R. 2021, *Kernels (Filters) in Convolutional Neural Network (CNN), Lets Talk About Them*, Viewed 7th Dec 2023, < https://medium.com/codex/kernels-filters-in-convolutional-neural-network-cnn-lets-talk-about-them-ee4e94f3319 >

SuperDataScience Team 2018, *Convolutional Neural Networks (CNN): Step 3 – Flattening*, Viewed 7th Dec 2023,< https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-3-flattening >

Ioffe S., Szegedy C, 2015, *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*, Viewed Dec 10th 2023, < https://arxiv.org/abs/1502.03167v3>

Hinton G., Srivastava N., Krizhevsky A., Sutskever I., Salakhutdinov R. 2012, *Improving Neural Networks by Preventing Co-Adaptation of Feature Detection*, Viewed Dec 10th 2023,< https://arxiv.org/abs/1207.0580 >