Setup

```
import random
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sn
```

Keras

```
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D,
AvgPool2D, BatchNormalization, Reshape, Resizing
from tensorflow.keras.applications import VGG16
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import LearningRateScheduler
```

Additional

```
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from keras.datasets import mnist
from keras.utils import to_categorical
```

Data Loading

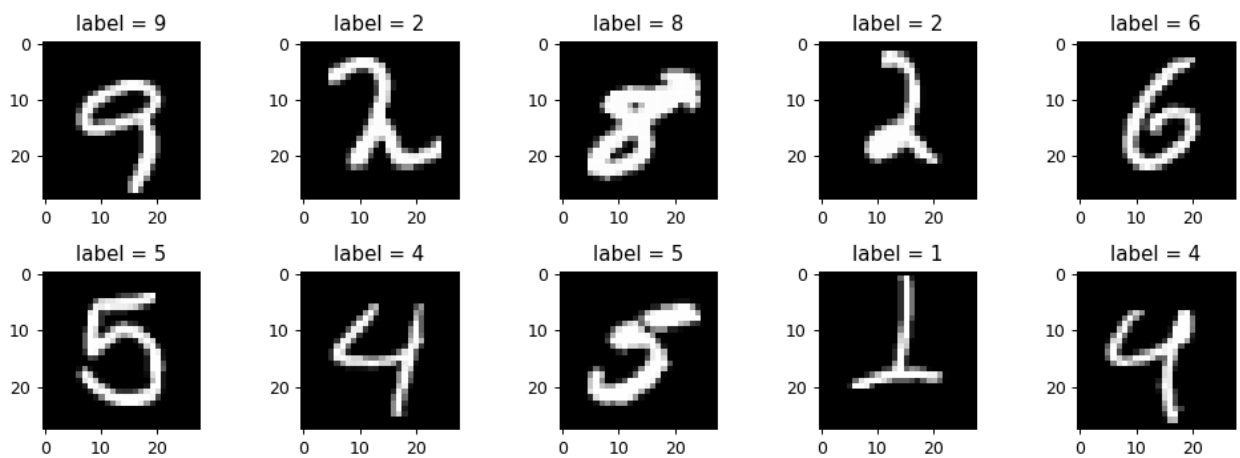
```
from keras.datasets import mnist


# load dataset
(x_train, y_train),(x_test, y_test) = mnist.load_data()



# count the number of unique train labels
unique, counts = np.unique(y_train, return_counts=True)
print("Train labels: ", dict(zip(unique, counts)))



# count the number of unique test labels
```

```
unique, counts = np.unique(y_test, return_counts=True)
print("\nTest labels: ", dict(zip(unique, counts)))
```

```
Train labels:  {0: 5923, 1: 6742, 2: 5958, 3: 6131, 4: 5842, 5: 5421,
6: 5918, 7: 6265, 8: 5851, 9: 5949}

Test labels:  {0: 980, 1: 1135, 2: 1032, 3: 1010, 4: 982, 5: 892, 6:
958, 7: 1028, 8: 974, 9: 1009}
```

Visualization

```
fig, axes = plt.subplots(nrows=2, ncols=5, figsize=(11, 4), dpi=90)
indxs = np.random.randint(0, x_train.shape[0], 10)

for i in range(2):
    for j in range(5) :
        axes[i, j].imshow(x_train[indxs[i*5 + j]], cmap='gray')
        axes[i, j].set_title(f'label = {y_train[indxs[i*5 + j]]}')

fig.tight_layout()
plt.show()
```




```
# convert to one-hot vector
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

image_size = x_train.shape[1]
input_size = image_size * image_size
input_size
```

```
784
```

```
# resize and normalize
# x_train = np.reshape(x_train, [-1, input_size])
x_train = x_train.astype('float32') / 255
```

```
# x_test = np.reshape(x_test, [-1, input_size])
x_test = x_test.astype('float32') / 255
```

Model Structure - CNN

CONFIG

```
EPOCHS = 10
BATCH_SIZE = 128

filter_sizes = [(3, 3), (5, 5)]
filter_nums = [32, 64]
```

Flatten or what?

After feature extraction by CNN we have to give a 1-D vector to our Dense (fully-conncted) layers for classification task, however it is important how we convert the features into a 1-D vectors.

There might be several ways but the two most common ways are using **`Flatten()`** by turning the whole `(w, h, d)` into `w*h*d` and giving it to the Dense layers.

Another way is using ***`GlobalMaxPooling()`*** which in each pixel takes the maximum of all the values through *depth*, so it takes `(w, h, d)` and gives us a `(w, h)` then we can Flatten() to get the final dimension of `w*d`.

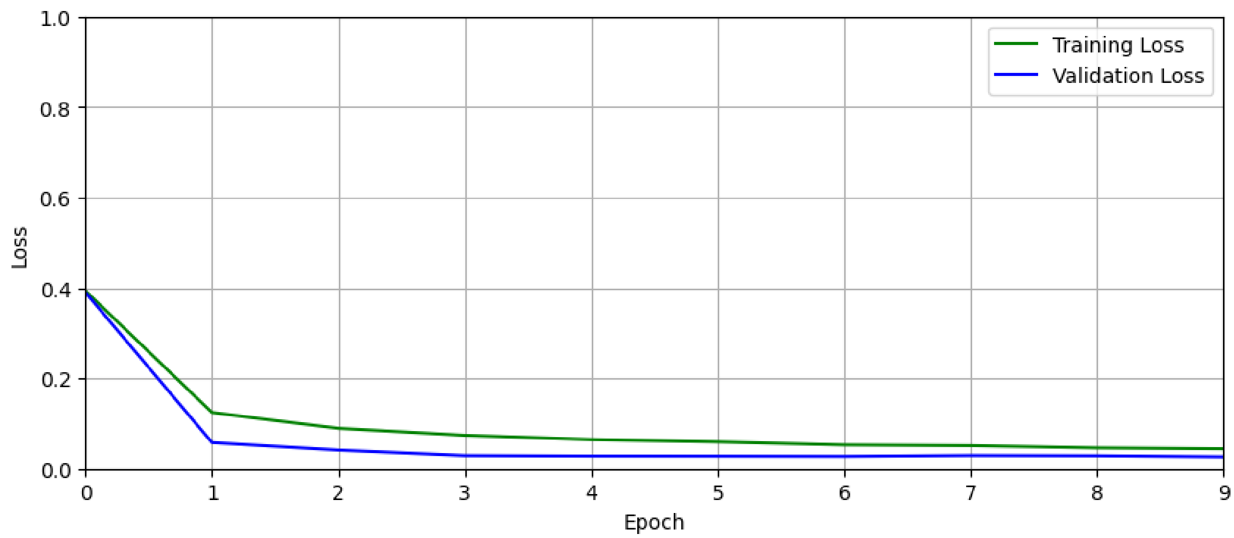
In this report we used the first method.

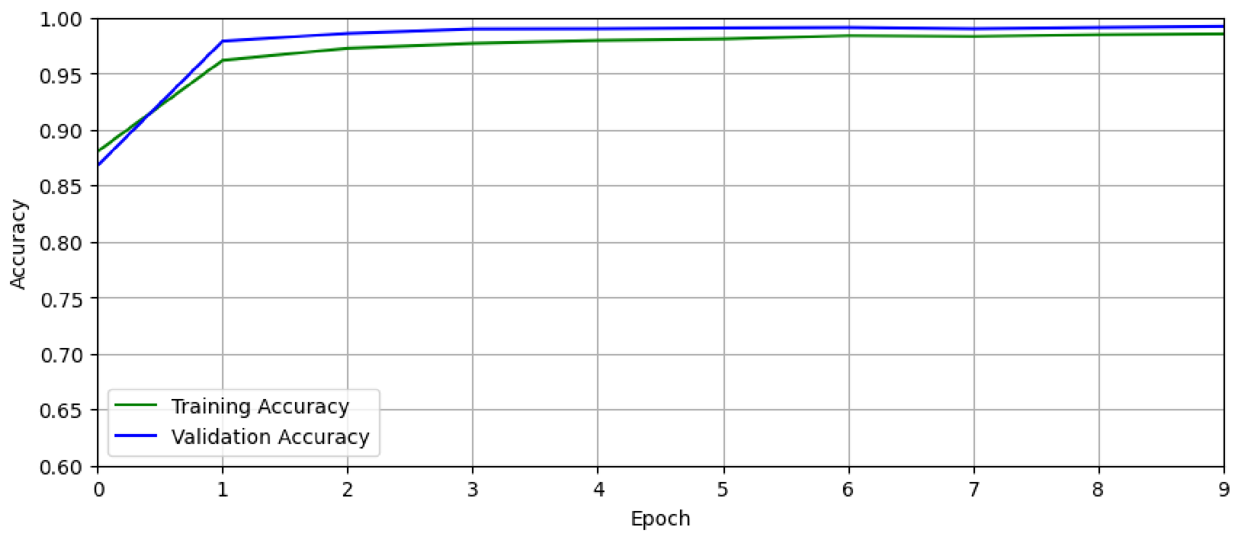
```
for sz in filter_sizes :
    for cnt in filter_nums :

        model = Sequential()
        # model.add(Resizing(32, 32, input_shape = (28, 28, 3)))

        model.add(Conv2D(32, (3, 3), activation = 'relu', input_shape
= (28, 28, 1)))
        model.add(BatchNormalization())
        model.add(MaxPool2D(strides = 2))
        model.add(Dropout(0.5))

        model.add(Conv2D(cnt, sz, activation = 'relu'))
        model.add(BatchNormalization())
        model.add(MaxPool2D(strides = 2))
        model.add(Dropout(0.5))

        model.add(Flatten())

        model.add(Dense(128, activation = 'relu'))
        model.add(Dense(10, activation = 'softmax'))
```

```
        model.summary()

        model.compile(optimizer='adam',
loss='categorical_crossentropy', metrics = ['accuracy'])

        history = model.fit(x_train, y_train,
                            validation_data=(x_test, y_test),
                            epochs = EPOCHS,
                            batch_size = BATCH_SIZE
                            )

        train_loss = history.history["loss"]
        train_acc  = history.history["accuracy"]
        valid_loss = history.history["val_loss"]
        valid_acc  = history.history["val_accuracy"]

        plot_results([ train_loss, valid_loss ],
                    ylabel="Loss",
                    ylim = [0.0, 1.0],
                    metric_name=["Training Loss", "Validation Loss"],
                    color=["g", "b"]);

        plot_results([ train_acc, valid_acc ],
                    ylabel="Accuracy",
                    ylim = [0.6, 1.0],
                    metric_name=["Training Accuracy", "Validation
Accuracy"],
                    color=["g", "b"])
```

```
Model: "sequential_17"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_19 (Conv2D)          (None, 26, 26, 32)        320       
                                                                 
 batch_normalization_17 (Ba  (None, 26, 26, 32)        128       
 tchNormalization)                                               
                                                                 
 max_pooling2d_17 (MaxPooli  (None, 13, 13, 32)        0         
 ng2D)                                                           
                                                                 
 dropout_23 (Dropout)        (None, 13, 13, 32)        0         
                                                                 
 conv2d_20 (Conv2D)          (None, 11, 11, 32)        9248      
                                                                 
 batch_normalization_18 (Ba  (None, 11, 11, 32)        128       
 tchNormalization)                                               
```

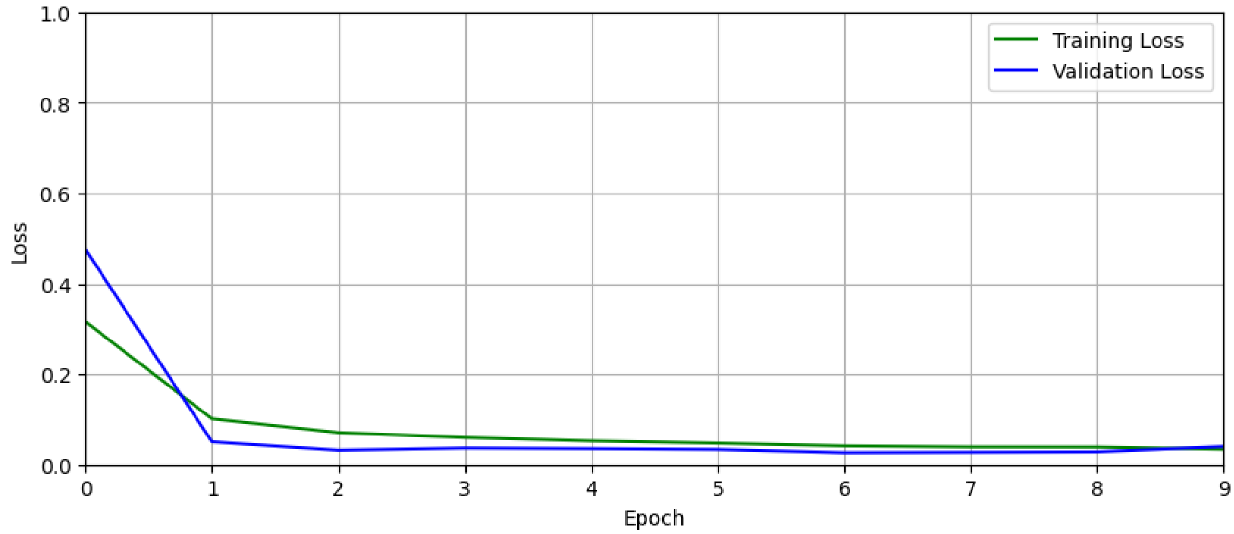
```
 max_pooling2d_18 (MaxPooli  (None, 5, 5, 32)          0
 ng2D)

 dropout_24 (Dropout)        (None, 5, 5, 32)          0

 flatten_8 (Flatten)         (None, 800)               0

 dense_27 (Dense)            (None, 128)               102528

 dense_28 (Dense)            (None, 10)                1290

=================================================================
Total params: 113642 (443.91 KB)
Trainable params: 113514 (443.41 KB)
Non-trainable params: 128 (512.00 Byte)
_________________________________________________________________
Epoch 1/10
469/469 [==============================] - 6s 8ms/step - loss: 0.3935
- accuracy: 0.8801 - val_loss: 0.3901 - val_accuracy: 0.8674
Epoch 2/10
469/469 [==============================] - 3s 6ms/step - loss: 0.1244
- accuracy: 0.9616 - val_loss: 0.0581 - val_accuracy: 0.9787
Epoch 3/10
469/469 [==============================] - 3s 6ms/step - loss: 0.0895
- accuracy: 0.9725 - val_loss: 0.0414 - val_accuracy: 0.9858
Epoch 4/10
469/469 [==============================] - 3s 7ms/step - loss: 0.0738
- accuracy: 0.9768 - val_loss: 0.0292 - val_accuracy: 0.9897
Epoch 5/10
469/469 [==============================] - 3s 7ms/step - loss: 0.0646
- accuracy: 0.9793 - val_loss: 0.0282 - val_accuracy: 0.9899
Epoch 6/10
469/469 [==============================] - 3s 7ms/step - loss: 0.0597
- accuracy: 0.9807 - val_loss: 0.0279 - val_accuracy: 0.9906
Epoch 7/10
469/469 [==============================] - 4s 8ms/step - loss: 0.0530
- accuracy: 0.9837 - val_loss: 0.0275 - val_accuracy: 0.9910
Epoch 8/10
469/469 [==============================] - 4s 8ms/step - loss: 0.0513
- accuracy: 0.9830 - val_loss: 0.0295 - val_accuracy: 0.9899
Epoch 9/10
469/469 [==============================] - 3s 7ms/step - loss: 0.0463
- accuracy: 0.9846 - val_loss: 0.0288 - val_accuracy: 0.9910
Epoch 10/10
469/469 [==============================] - 3s 7ms/step - loss: 0.0443
- accuracy: 0.9854 - val_loss: 0.0265 - val_accuracy: 0.9920
```

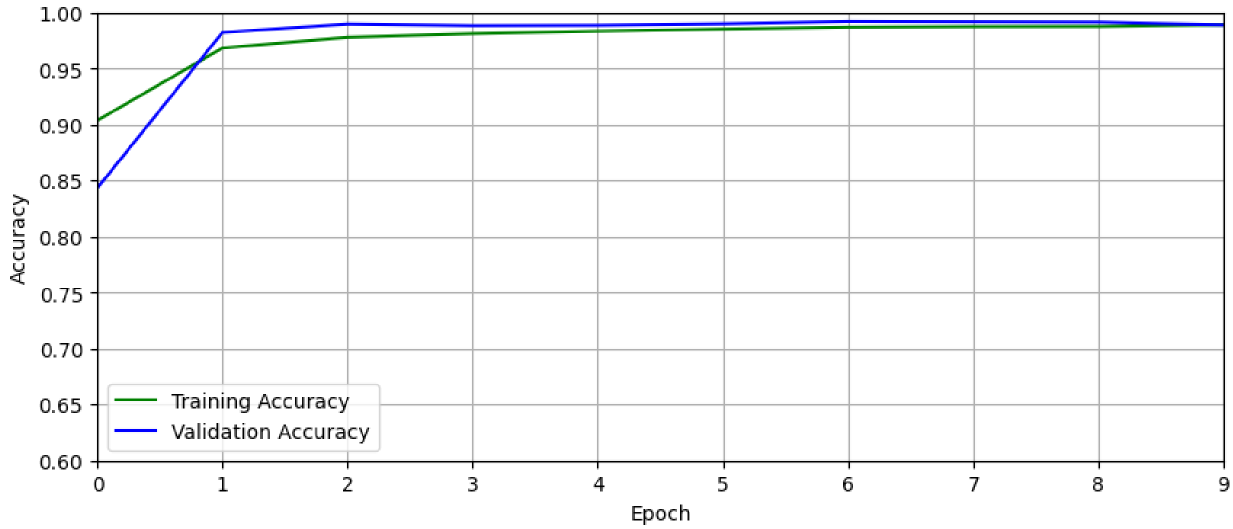





```
Model: "sequential_18"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_21 (Conv2D)          (None, 26, 26, 32)        320

 batch_normalization_19 (Ba  (None, 26, 26, 32)        128
 tchNormalization)

 max_pooling2d_19 (MaxPooli  (None, 13, 13, 32)        0
 ng2D)

 dropout_25 (Dropout)        (None, 13, 13, 32)        0

 conv2d_22 (Conv2D)          (None, 11, 11, 64)        18496
```

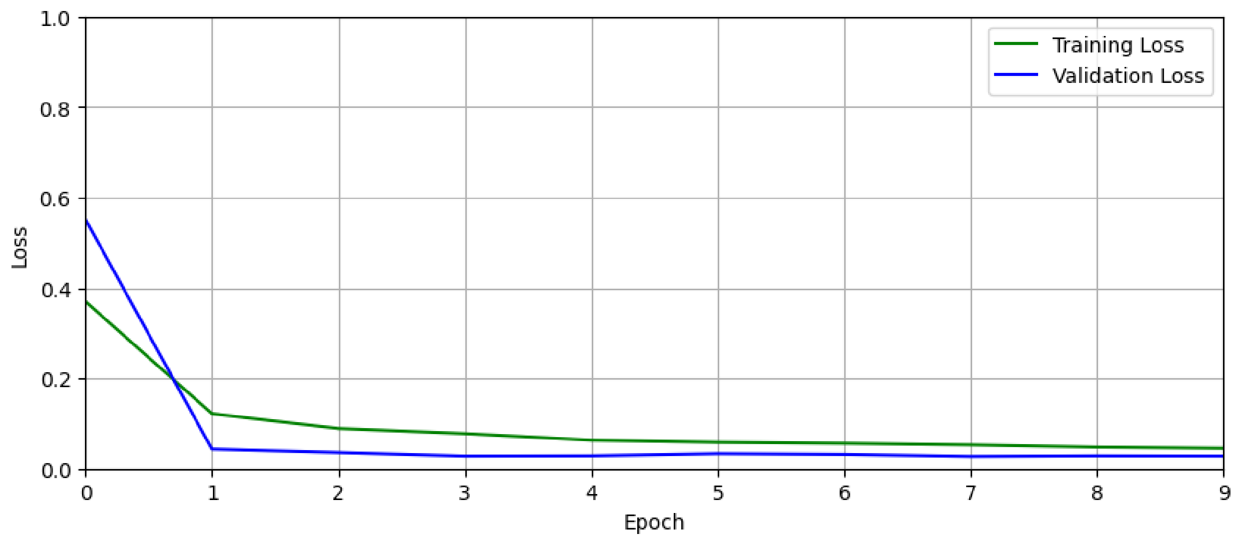
```
 batch_normalization_20 (Ba  (None, 11, 11, 64)        256
 tchNormalization)

 max_pooling2d_20 (MaxPooli  (None, 5, 5, 64)          0
 ng2D)

 dropout_26 (Dropout)        (None, 5, 5, 64)          0

 flatten_9 (Flatten)         (None, 1600)              0

 dense_29 (Dense)            (None, 128)               204928

 dense_30 (Dense)            (None, 10)                1290

=================================================================
Total params: 225418 (880.54 KB)
Trainable params: 225226 (879.79 KB)
Non-trainable params: 192 (768.00 Byte)
_________________________________________________________________
Epoch 1/10
469/469 [==============================] - 5s 7ms/step - loss: 0.3160
- accuracy: 0.9035 - val_loss: 0.4756 - val_accuracy: 0.8431
Epoch 2/10
469/469 [==============================] - 4s 8ms/step - loss: 0.1016
- accuracy: 0.9682 - val_loss: 0.0507 - val_accuracy: 0.9821
Epoch 3/10
469/469 [==============================] - 3s 7ms/step - loss: 0.0707
- accuracy: 0.9778 - val_loss: 0.0321 - val_accuracy: 0.9896
Epoch 4/10
469/469 [==============================] - 3s 7ms/step - loss: 0.0604
- accuracy: 0.9810 - val_loss: 0.0368 - val_accuracy: 0.9882
Epoch 5/10
469/469 [==============================] - 4s 7ms/step - loss: 0.0528
- accuracy: 0.9833 - val_loss: 0.0356 - val_accuracy: 0.9885
Epoch 6/10
469/469 [==============================] - 3s 7ms/step - loss: 0.0477
- accuracy: 0.9851 - val_loss: 0.0338 - val_accuracy: 0.9898
Epoch 7/10
469/469 [==============================] - 3s 7ms/step - loss: 0.0417
- accuracy: 0.9869 - val_loss: 0.0266 - val_accuracy: 0.9919
Epoch 8/10
469/469 [==============================] - 3s 7ms/step - loss: 0.0395
- accuracy: 0.9873 - val_loss: 0.0275 - val_accuracy: 0.9916
Epoch 9/10
469/469 [==============================] - 4s 8ms/step - loss: 0.0392
- accuracy: 0.9875 - val_loss: 0.0284 - val_accuracy: 0.9913
Epoch 10/10
469/469 [==============================] - 3s 7ms/step - loss: 0.0342
- accuracy: 0.9890 - val_loss: 0.0404 - val_accuracy: 0.9888
```

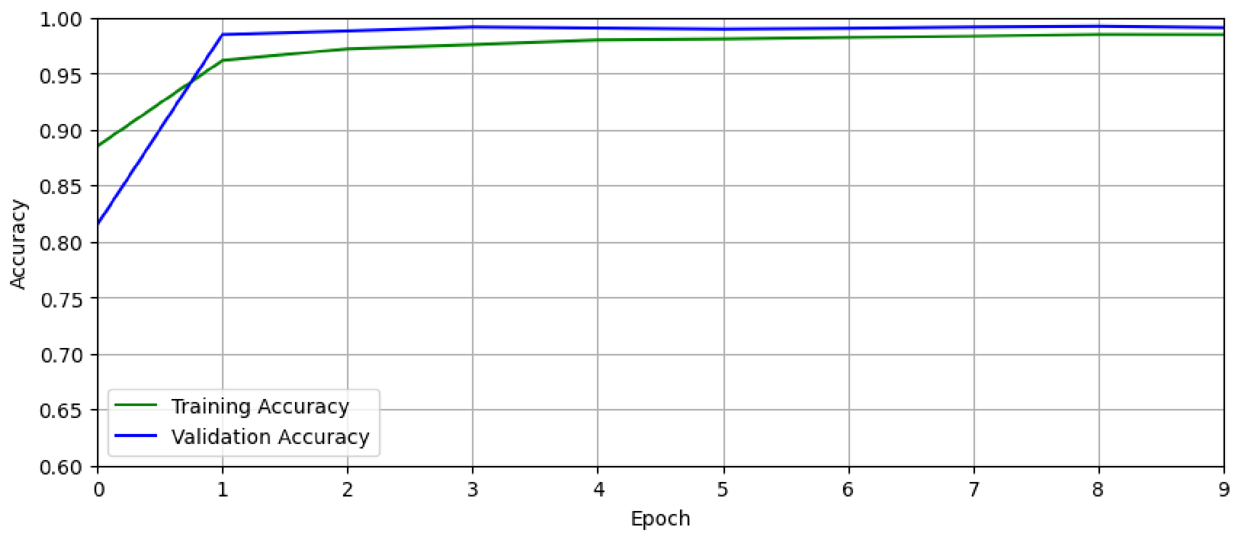





```
Model: "sequential_19"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_23 (Conv2D)          (None, 26, 26, 32)        320

 batch_normalization_21 (Ba  (None, 26, 26, 32)        128
 tchNormalization)

 max_pooling2d_21 (MaxPooli  (None, 13, 13, 32)        0
 ng2D)

 dropout_27 (Dropout)        (None, 13, 13, 32)        0

 conv2d_24 (Conv2D)          (None, 9, 9, 32)          25632
```

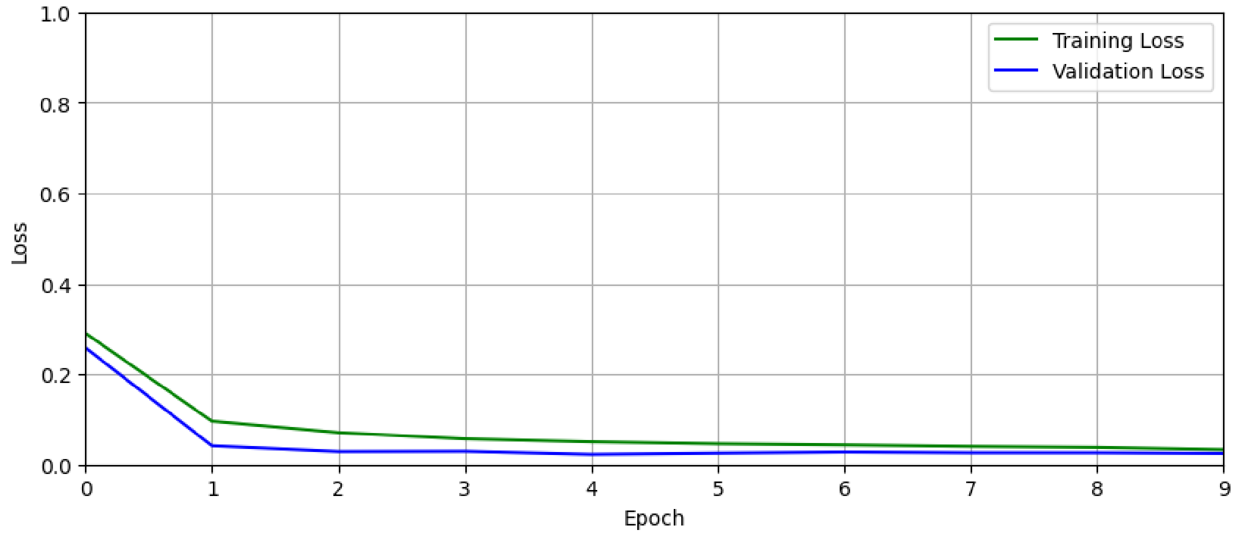
```
 batch_normalization_22 (Ba  (None, 9, 9, 32)          128
 tchNormalization)

 max_pooling2d_22 (MaxPooli  (None, 4, 4, 32)          0
 ng2D)

 dropout_28 (Dropout)        (None, 4, 4, 32)          0

 flatten_10 (Flatten)        (None, 512)               0

 dense_31 (Dense)            (None, 128)               65664

 dense_32 (Dense)            (None, 10)                1290

=================================================================
Total params: 93162 (363.91 KB)
Trainable params: 93034 (363.41 KB)
Non-trainable params: 128 (512.00 Byte)
_________________________________________________________________
Epoch 1/10
469/469 [==============================] - 6s 8ms/step - loss: 0.3701
- accuracy: 0.8850 - val_loss: 0.5515 - val_accuracy: 0.8149
Epoch 2/10
469/469 [==============================] - 3s 7ms/step - loss: 0.1219
- accuracy: 0.9616 - val_loss: 0.0435 - val_accuracy: 0.9848
Epoch 3/10
469/469 [==============================] - 3s 7ms/step - loss: 0.0891
- accuracy: 0.9720 - val_loss: 0.0360 - val_accuracy: 0.9880
Epoch 4/10
469/469 [==============================] - 3s 7ms/step - loss: 0.0777
- accuracy: 0.9757 - val_loss: 0.0281 - val_accuracy: 0.9914
Epoch 5/10
469/469 [==============================] - 3s 7ms/step - loss: 0.0633
- accuracy: 0.9798 - val_loss: 0.0287 - val_accuracy: 0.9905
Epoch 6/10
469/469 [==============================] - 3s 6ms/step - loss: 0.0587
- accuracy: 0.9807 - val_loss: 0.0335 - val_accuracy: 0.9896
Epoch 7/10
469/469 [==============================] - 3s 7ms/step - loss: 0.0563
- accuracy: 0.9821 - val_loss: 0.0318 - val_accuracy: 0.9903
Epoch 8/10
469/469 [==============================] - 3s 7ms/step - loss: 0.0530
- accuracy: 0.9833 - val_loss: 0.0274 - val_accuracy: 0.9914
Epoch 9/10
469/469 [==============================] - 3s 7ms/step - loss: 0.0479
- accuracy: 0.9848 - val_loss: 0.0287 - val_accuracy: 0.9921
Epoch 10/10
469/469 [==============================] - 3s 7ms/step - loss: 0.0454
- accuracy: 0.9847 - val_loss: 0.0281 - val_accuracy: 0.9909
```

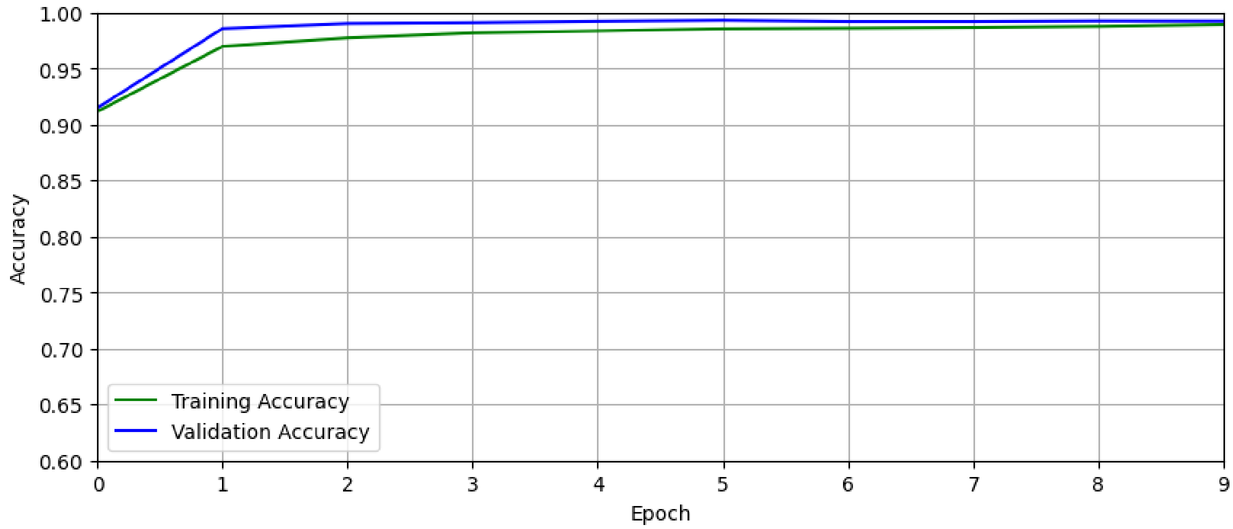





```
Model: "sequential_20"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_25 (Conv2D)          (None, 26, 26, 32)        320

 batch_normalization_23 (Ba  (None, 26, 26, 32)        128
 tchNormalization)

 max_pooling2d_23 (MaxPooli  (None, 13, 13, 32)        0
 ng2D)

 dropout_29 (Dropout)        (None, 13, 13, 32)        0

 conv2d_26 (Conv2D)          (None, 9, 9, 64)          51264
```

```
 batch_normalization_24 (Ba  (None, 9, 9, 64)          256
 tchNormalization)

 max_pooling2d_24 (MaxPooli  (None, 4, 4, 64)          0
 ng2D)

 dropout_30 (Dropout)        (None, 4, 4, 64)          0

 flatten_11 (Flatten)        (None, 1024)              0

 dense_33 (Dense)            (None, 128)               131200

 dense_34 (Dense)            (None, 10)                1290

=================================================================
Total params: 184458 (720.54 KB)
Trainable params: 184266 (719.79 KB)
Non-trainable params: 192 (768.00 Byte)
_________________________________________________________________
Epoch 1/10
469/469 [==============================] - 6s 9ms/step - loss: 0.2906
- accuracy: 0.9115 - val_loss: 0.2584 - val_accuracy: 0.9147
Epoch 2/10
469/469 [==============================] - 3s 7ms/step - loss: 0.0962
- accuracy: 0.9696 - val_loss: 0.0420 - val_accuracy: 0.9856
Epoch 3/10
469/469 [==============================] - 3s 7ms/step - loss: 0.0709
- accuracy: 0.9773 - val_loss: 0.0293 - val_accuracy: 0.9901
Epoch 4/10
469/469 [==============================] - 3s 7ms/step - loss: 0.0573
- accuracy: 0.9817 - val_loss: 0.0299 - val_accuracy: 0.9908
Epoch 5/10
469/469 [==============================] - 4s 8ms/step - loss: 0.0507
- accuracy: 0.9835 - val_loss: 0.0231 - val_accuracy: 0.9920
Epoch 6/10
469/469 [==============================] - 3s 7ms/step - loss: 0.0464
- accuracy: 0.9855 - val_loss: 0.0259 - val_accuracy: 0.9929
Epoch 7/10
469/469 [==============================] - 3s 7ms/step - loss: 0.0438
- accuracy: 0.9860 - val_loss: 0.0283 - val_accuracy: 0.9918
Epoch 8/10
469/469 [==============================] - 4s 8ms/step - loss: 0.0405
- accuracy: 0.9866 - val_loss: 0.0267 - val_accuracy: 0.9918
Epoch 9/10
469/469 [==============================] - 3s 7ms/step - loss: 0.0384
- accuracy: 0.9876 - val_loss: 0.0267 - val_accuracy: 0.9923
Epoch 10/10
469/469 [==============================] - 3s 7ms/step - loss: 0.0334
- accuracy: 0.9892 - val_loss: 0.0253 - val_accuracy: 0.9922
```





Visualize The results

```
def plot_results(metrics, title=None, ylabel=None, ylim=None,
metric_name=None, color=None):

    fig, ax = plt.subplots(figsize=(10, 4))

    if not (isinstance(metric_name, list) or isinstance(metric_name,
tuple)):
        metrics = [metrics,]
        metric_name = [metric_name,]

    for idx, metric in enumerate(metrics):
        ax.plot(metric, color=color[idx])
```

```
    plt.xlabel("Epoch")
    plt.ylabel(ylabel)
    plt.title(title)
    plt.xlim([0, EPOCHS-1])
    plt.ylim(ylim)
    # Tailor x-axis tick marks
    plt.grid(True)
    plt.legend(metric_name)
    plt.show()
    plt.close()
```

Results

As we can see, we have achieved the accuracy of $99.22\,\%$ and loss of 0.0253 with `kernel_size = (5, 5)` and `filters = 64` which is out best result so far.

Dropout Yes or No?

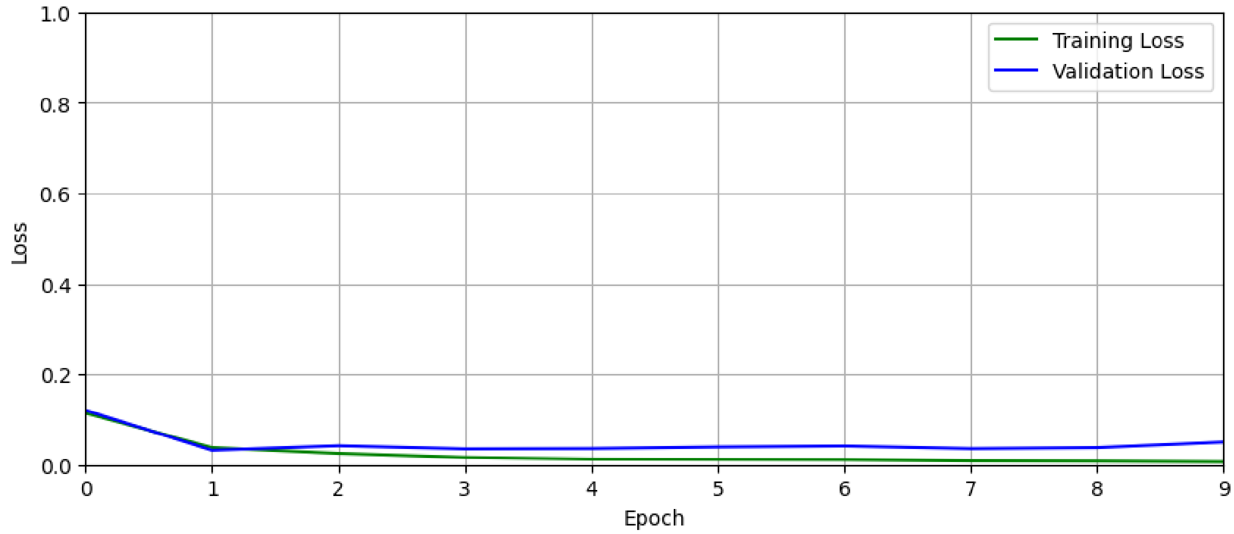
Without Dropout

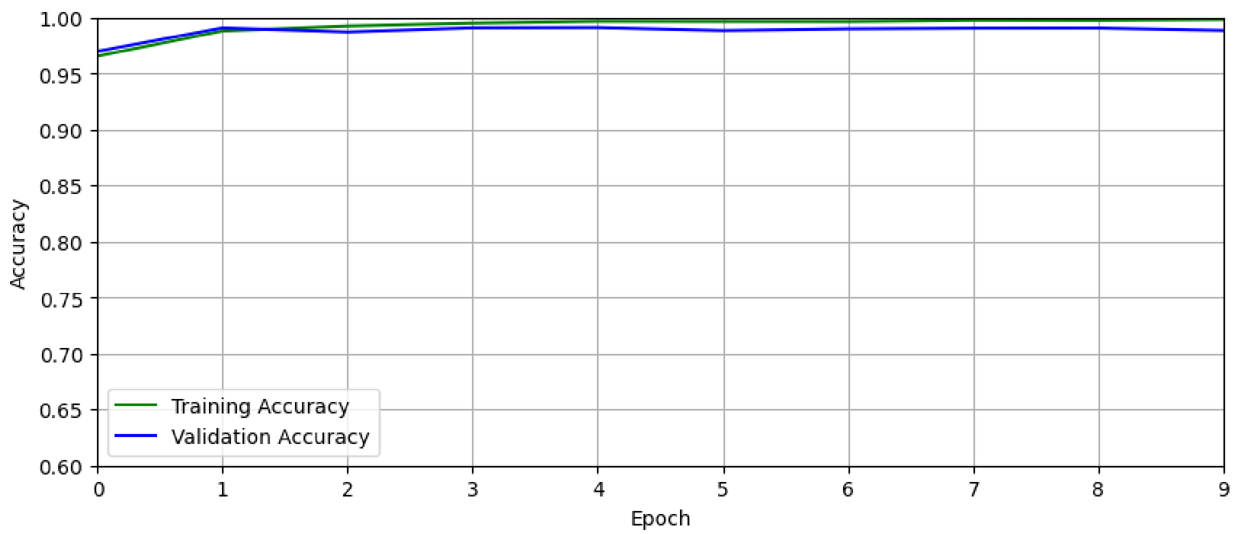
```
model = Sequential()

model.add(Conv2D(32, (3, 3), activation = 'relu', input_shape = (28,
28, 1)))
model.add(BatchNormalization())
model.add(MaxPool2D(strides = 2))
# model.add(Dropout(0.5))

model.add(Conv2D(64, (5, 5), activation = 'relu'))
model.add(BatchNormalization())
model.add(MaxPool2D(strides = 2))
# model.add(Dropout(0.5))

model.add(Flatten())

model.add(Dense(128, activation = 'relu'))
model.add(Dense(10, activation = 'softmax'))

model.summary()

model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics = ['accuracy'])

history = model.fit(x_train, y_train,
    validation_data=(x_test, y_test),
    epochs = EPOCHS,
```

```
    batch_size = BATCH_SIZE
)

train_loss = history.history["loss"]
train_acc  = history.history["accuracy"]
valid_loss = history.history["val_loss"]
valid_acc  = history.history["val_accuracy"]

plot_results([ train_loss, valid_loss ],
    ylabel="Loss",
    ylim = [0.0, 1.0],
    metric_name=["Training Loss", "Validation Loss"],
    color=["g", "b"]
);

plot_results([ train_acc, valid_acc ],
    ylabel="Accuracy",
    ylim = [0.6, 1.0],
    metric_name=["Training Accuracy", "Validation Accuracy"],
    color=["g", "b"]
);
```

```
Model: "sequential_21"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_27 (Conv2D)          (None, 26, 26, 32)        320

 batch_normalization_25 (Ba  (None, 26, 26, 32)        128
 tchNormalization)

 max_pooling2d_25 (MaxPooli  (None, 13, 13, 32)        0
 ng2D)

 conv2d_28 (Conv2D)          (None, 9, 9, 64)          51264

 batch_normalization_26 (Ba  (None, 9, 9, 64)          256
 tchNormalization)

 max_pooling2d_26 (MaxPooli  (None, 4, 4, 64)          0
 ng2D)

 flatten_12 (Flatten)        (None, 1024)              0

 dense_35 (Dense)            (None, 128)               131200

 dense_36 (Dense)            (None, 10)                1290

=================================================================
Total params: 184458 (720.54 KB)
```

```
Trainable params: 184266 (719.79 KB)
Non-trainable params: 192 (768.00 Byte)
_________________________________________________________________
Epoch 1/10
469/469 [==============================] - 5s 7ms/step - loss: 0.1142
- accuracy: 0.9654 - val_loss: 0.1204 - val_accuracy: 0.9697
Epoch 2/10
469/469 [==============================] - 3s 6ms/step - loss: 0.0380
- accuracy: 0.9879 - val_loss: 0.0321 - val_accuracy: 0.9906
Epoch 3/10
469/469 [==============================] - 3s 6ms/step - loss: 0.0250
- accuracy: 0.9922 - val_loss: 0.0420 - val_accuracy: 0.9870
Epoch 4/10
469/469 [==============================] - 3s 6ms/step - loss: 0.0161
- accuracy: 0.9948 - val_loss: 0.0351 - val_accuracy: 0.9907
Epoch 5/10
469/469 [==============================] - 3s 6ms/step - loss: 0.0122
- accuracy: 0.9964 - val_loss: 0.0358 - val_accuracy: 0.9909
Epoch 6/10
469/469 [==============================] - 3s 6ms/step - loss: 0.0117
- accuracy: 0.9963 - val_loss: 0.0393 - val_accuracy: 0.9883
Epoch 7/10
469/469 [==============================] - 3s 6ms/step - loss: 0.0112
- accuracy: 0.9962 - val_loss: 0.0414 - val_accuracy: 0.9898
Epoch 8/10
469/469 [==============================] - 3s 6ms/step - loss: 0.0086
- accuracy: 0.9973 - val_loss: 0.0356 - val_accuracy: 0.9904
Epoch 9/10
469/469 [==============================] - 3s 7ms/step - loss: 0.0080
- accuracy: 0.9972 - val_loss: 0.0378 - val_accuracy: 0.9905
Epoch 10/10
469/469 [==============================] - 3s 6ms/step - loss: 0.0066
- accuracy: 0.9979 - val_loss: 0.0500 - val_accuracy: 0.9884
```





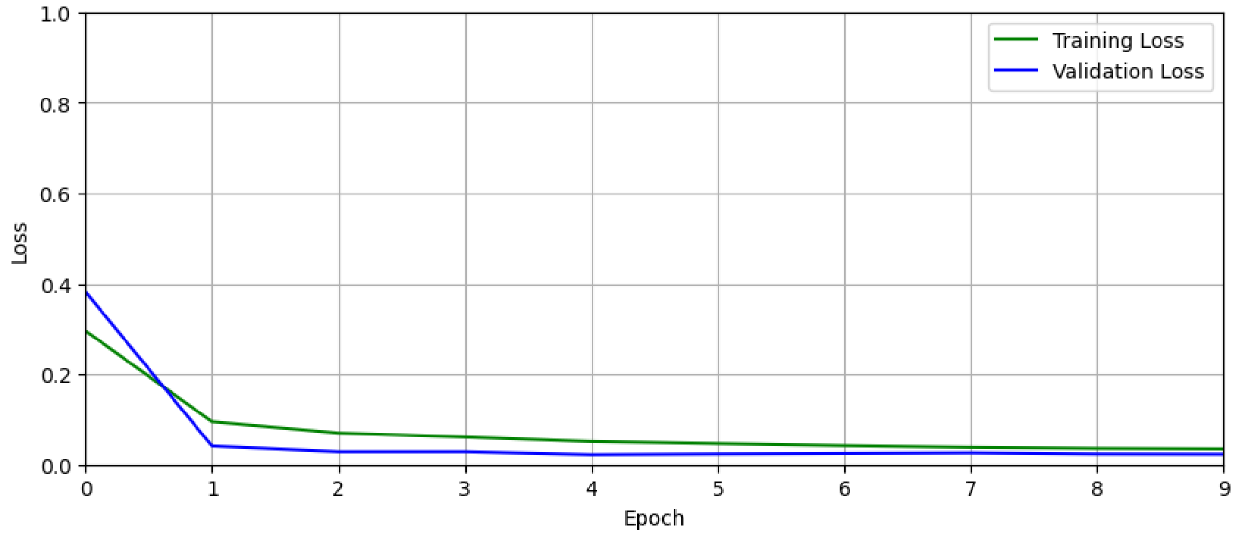
With Dropout

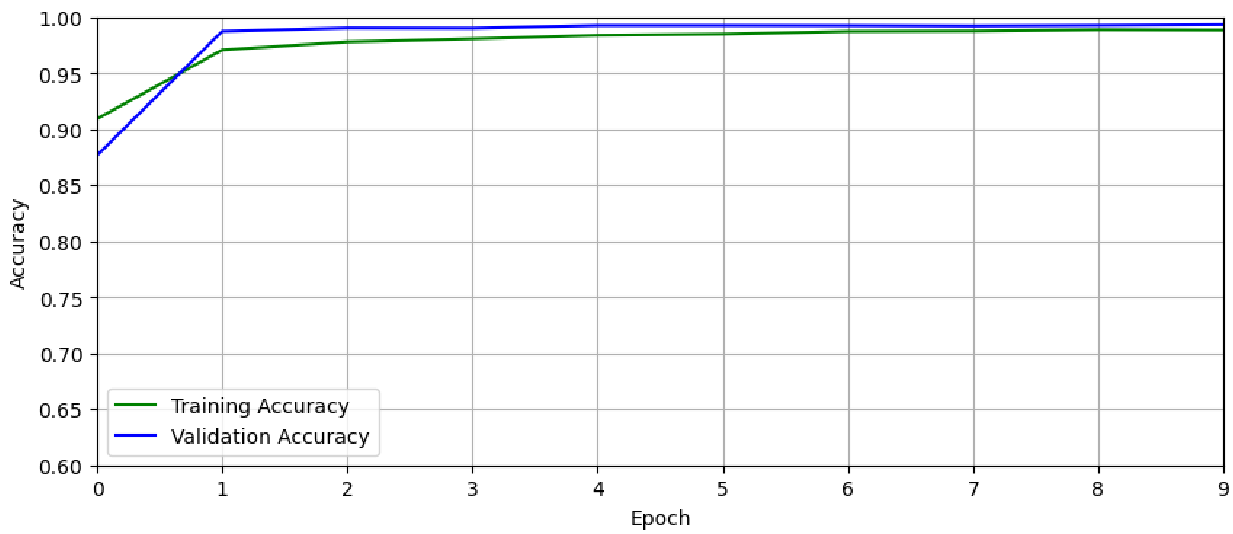
```
model = Sequential()

model.add(Conv2D(32, (3, 3), activation = 'relu', input_shape = (28, 
28, 1)))
model.add(BatchNormalization())
model.add(MaxPool2D(strides = 2))
model.add(Dropout(0.5))

model.add(Conv2D(64, (5, 5), activation = 'relu'))
model.add(BatchNormalization())
model.add(MaxPool2D(strides = 2))
model.add(Dropout(0.5))

model.add(Flatten())
```

```
model.add(Dense(128, activation = 'relu'))
model.add(Dense(10, activation = 'softmax'))

model.summary()

model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics = ['accuracy'])

history = model.fit(x_train, y_train,
    validation_data=(x_test, y_test),
    epochs = EPOCHS,
    batch_size = BATCH_SIZE
)

train_loss = history.history["loss"]
train_acc  = history.history["accuracy"]
valid_loss = history.history["val_loss"]
valid_acc  = history.history["val_accuracy"]

plot_results([ train_loss, valid_loss ],
    ylabel="Loss",
    ylim = [0.0, 1.0],
    metric_name=["Training Loss", "Validation Loss"],
    color=["g", "b"]
);

plot_results([ train_acc, valid_acc ],
    ylabel="Accuracy",
    ylim = [0.6, 1.0],
    metric_name=["Training Accuracy", "Validation Accuracy"],
    color=["g", "b"]
);
```

```
Model: "sequential_22"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_29 (Conv2D)          (None, 26, 26, 32)        320

 batch_normalization_27 (Ba  (None, 26, 26, 32)        128
 tchNormalization)

 max_pooling2d_27 (MaxPooli  (None, 13, 13, 32)        0
 ng2D)

 dropout_31 (Dropout)        (None, 13, 13, 32)        0

 conv2d_30 (Conv2D)          (None, 9, 9, 64)          51264
```

```
 batch_normalization_28 (Ba  (None, 9, 9, 64)          256
 tchNormalization)

 max_pooling2d_28 (MaxPooli  (None, 4, 4, 64)          0
 ng2D)

 dropout_32 (Dropout)        (None, 4, 4, 64)          0

 flatten_13 (Flatten)        (None, 1024)              0

 dense_37 (Dense)            (None, 128)               131200

 dense_38 (Dense)            (None, 10)                1290

=================================================================
Total params: 184458 (720.54 KB)
Trainable params: 184266 (719.79 KB)
Non-trainable params: 192 (768.00 Byte)
_________________________________________________________________
Epoch 1/10
469/469 [==============================] - 5s 8ms/step - loss: 0.2958
- accuracy: 0.9092 - val_loss: 0.3821 - val_accuracy: 0.8766
Epoch 2/10
469/469 [==============================] - 4s 8ms/step - loss: 0.0951
- accuracy: 0.9707 - val_loss: 0.0415 - val_accuracy: 0.9874
Epoch 3/10
469/469 [==============================] - 3s 7ms/step - loss: 0.0699
- accuracy: 0.9779 - val_loss: 0.0288 - val_accuracy: 0.9903
Epoch 4/10
469/469 [==============================] - 3s 7ms/step - loss: 0.0612
- accuracy: 0.9806 - val_loss: 0.0289 - val_accuracy: 0.9901
Epoch 5/10
469/469 [==============================] - 4s 8ms/step - loss: 0.0513
- accuracy: 0.9838 - val_loss: 0.0224 - val_accuracy: 0.9925
Epoch 6/10
469/469 [==============================] - 3s 7ms/step - loss: 0.0467
- accuracy: 0.9849 - val_loss: 0.0240 - val_accuracy: 0.9925
Epoch 7/10
469/469 [==============================] - 3s 7ms/step - loss: 0.0422
- accuracy: 0.9873 - val_loss: 0.0252 - val_accuracy: 0.9924
Epoch 8/10
469/469 [==============================] - 3s 7ms/step - loss: 0.0385
- accuracy: 0.9876 - val_loss: 0.0264 - val_accuracy: 0.9921
Epoch 9/10
469/469 [==============================] - 4s 8ms/step - loss: 0.0361
- accuracy: 0.9888 - val_loss: 0.0239 - val_accuracy: 0.9926
Epoch 10/10
469/469 [==============================] - 3s 7ms/step - loss: 0.0349
- accuracy: 0.9885 - val_loss: 0.0234 - val_accuracy: 0.9933
```



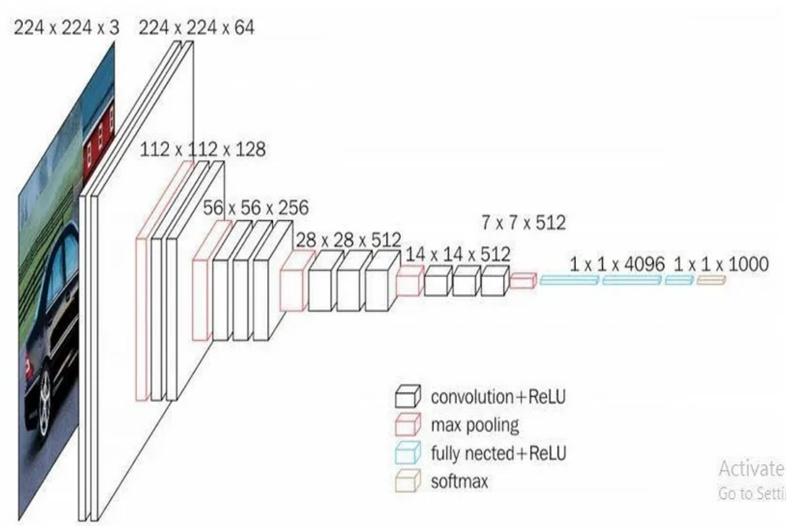


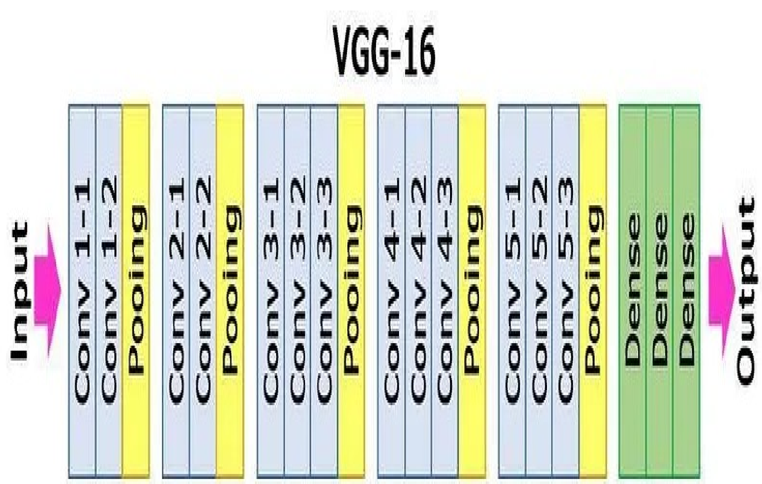
Results

Here it is obvious that by using Dropout not only we decrease the rate of **overfitting** and increased **generalization** but also the validation accuracy **with Dropout** is $99.33\,\%$ while it's only $98.84\,\%$ **without Dropout**

which means using Dropout can be useful.

VGG16





Structure

- The 16 in VGG16 refers to 16 layers that have weights. In VGG16 there are thirteen convolutional layers, five Max Pooling layers, and three Dense layers which sum up to 21 layers but it has only sixteen weight layers i.e., learnable parameters layer.
- VGG16 takes input tensor size as 224, 244 with 3 RGB channel
- Most unique thing about VGG16 is that instead of having a large number of hyper-parameters they focused on having convolution layers of 3x3 filter with stride 1 and always used the same padding and maxpool layer of 2x2 filter of stride 2.
- The convolution and max pool layers are consistently arranged throughout the whole architecture

- Conv-1 Layer has 64 number of filters, Conv-2 has 128 filters, Conv-3 has 256 filters, Conv 4 and Conv 5 has 512 filters.
- Three Fully-Connected (FC) layers follow a stack of convolutional layers: the first two have 4096 channels each, the third performs 1000-way ILSVRC classification and thus contains 1000 channels (one for each class). The final layer is the soft-max layer.

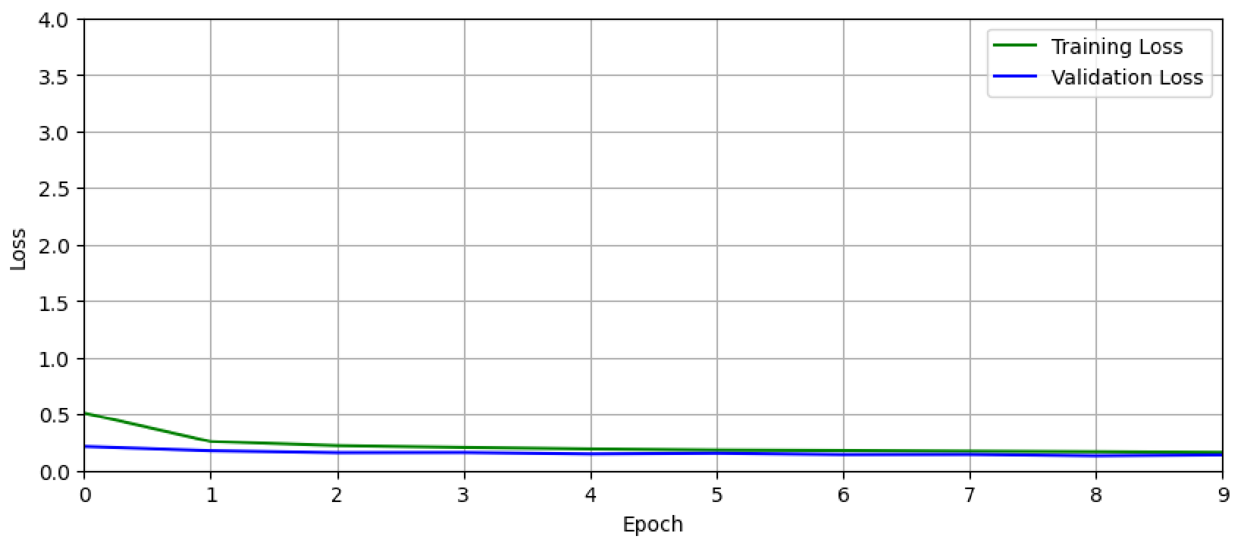
```
(x_train, y_train),(x_test, y_test) = mnist.load_data()

x_train = np.expand_dims(x_train, axis = -1)
x_test = np.expand_dims(x_test, axis = -1)

x_train = np.repeat(x_train, 3, axis = -1)
x_test = np.repeat(x_test, 3, axis = -1)

x_train = tf.image.resize(x_train, [32, 32])
x_test = tf.image.resize(x_test, [32, 32])

# convert to one-hot vector
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)


vgg_base = VGG16(weights = 'imagenet', include_top = False,
input_shape = (32, 32, 3))

vgg_base.trainable = False

model = Sequential()

model.add(vgg_base)

model.add(Flatten())

model.add(Dense(128, activation = 'relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(10, activation = 'softmax'))

model.summary()

Model: "sequential_24"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 1, 1, 512)         14714688

 flatten_15 (Flatten)        (None, 512)               0
```

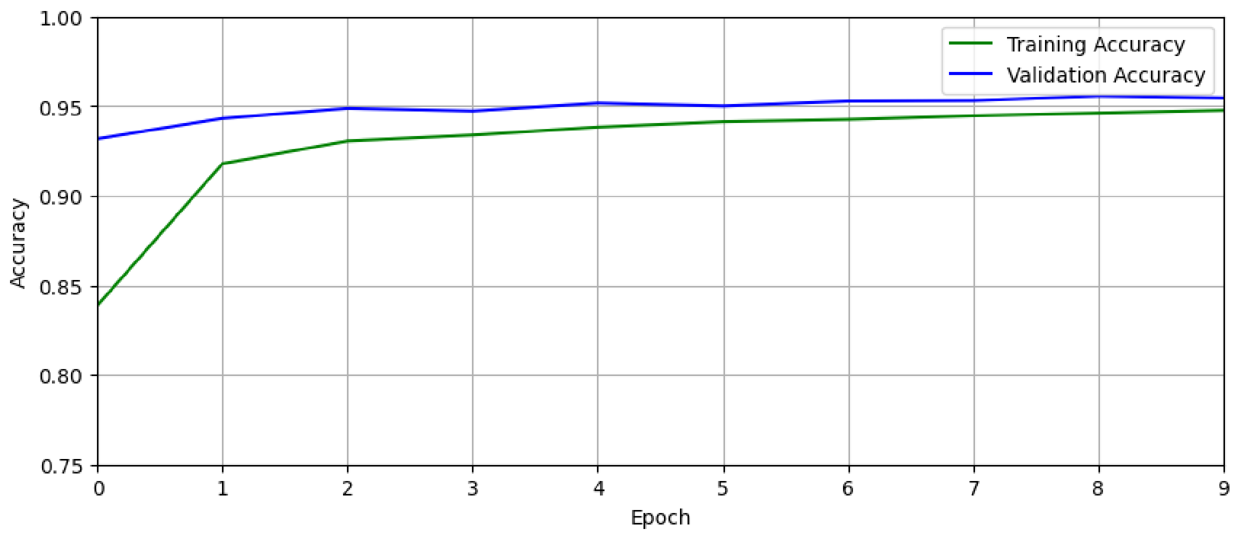
```
 dense_41 (Dense)            (None, 128)               65664

 batch_normalization_30 (Ba  (None, 128)               512
 tchNormalization)

 dropout_34 (Dropout)        (None, 128)               0

 dense_42 (Dense)            (None, 10)                1290

=================================================================
Total params: 14782154 (56.39 MB)
Trainable params: 67210 (262.54 KB)
Non-trainable params: 14714944 (56.13 MB)
_________________________________________________________________
```

```
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics = ['accuracy'])

history = model.fit(x_train, y_train,
    validation_data=(x_test, y_test),
    epochs = EPOCHS,
    batch_size = BATCH_SIZE
)

train_loss = history.history["loss"]
train_acc  = history.history["accuracy"]
valid_loss = history.history["val_loss"]
valid_acc  = history.history["val_accuracy"]

plot_results([ train_loss, valid_loss ],
    ylabel="Loss",
    ylim = [0.0, 0.4],
    metric_name=["Training Loss", "Validation Loss"],
    color=["g", "b"]
);

plot_results([ train_acc, valid_acc ],
    ylabel="Accuracy",
    ylim = [0.75, 1.0],
    metric_name=["Training Accuracy", "Validation Accuracy"],
    color=["g", "b"]
);
```

```
Epoch 1/10
469/469 [==============================] - 68s 23ms/step - loss:
0.5105 - accuracy: 0.8386 - val_loss: 0.2133 - val_accuracy: 0.9317
Epoch 2/10
469/469 [==============================] - 9s 19ms/step - loss: 0.2586
- accuracy: 0.9178 - val_loss: 0.1763 - val_accuracy: 0.9431
Epoch 3/10
```

```
469/469 [==============================] - 9s 19ms/step - loss: 0.2207
- accuracy: 0.9305 - val_loss: 0.1594 - val_accuracy: 0.9487
Epoch 4/10
469/469 [==============================] - 9s 19ms/step - loss: 0.2051
- accuracy: 0.9338 - val_loss: 0.1608 - val_accuracy: 0.9472
Epoch 5/10
469/469 [==============================] - 9s 19ms/step - loss: 0.1919
- accuracy: 0.9382 - val_loss: 0.1486 - val_accuracy: 0.9516
Epoch 6/10
469/469 [==============================] - 9s 19ms/step - loss: 0.1824
- accuracy: 0.9412 - val_loss: 0.1547 - val_accuracy: 0.9500
Epoch 7/10
469/469 [==============================] - 9s 19ms/step - loss: 0.1781
- accuracy: 0.9425 - val_loss: 0.1431 - val_accuracy: 0.9527
Epoch 8/10
469/469 [==============================] - 9s 19ms/step - loss: 0.1724
- accuracy: 0.9446 - val_loss: 0.1441 - val_accuracy: 0.9530
Epoch 9/10
469/469 [==============================] - 9s 19ms/step - loss: 0.1677
- accuracy: 0.9461 - val_loss: 0.1329 - val_accuracy: 0.9556
Epoch 10/10
469/469 [==============================] - 9s 19ms/step - loss: 0.1620
- accuracy: 0.9476 - val_loss: 0.1400 - val_accuracy: 0.9545
```





Results

At first glance the result might be unexpected due to the fact that we only were able to achieve an accuracy of $95.45\,\%$ on the validation set while we were able to achieve accuracy of more than $99\,\%$ in our own custom CNN model despite the fact that the VGG16 model is much stronger and was trained on a huge dataset like *Imagenet*.

The main idea and the key difference is perhaps that VGG16 model although is great and really powerful, it was not trained on this dataset but rather on a competely different dataset, also due to its deep structure the high-level and complex features that it has extracted, may not be useful for classifying hand-written digits dataset like MNIST while on the other hand our custom CNN has learned the somewhat high-level features specific to hand-written digits or as some would say our model was ***fine-tuned*** for this specific task.

What is the solution you may ask? ***Transfer Learning!***

Transfer Learning

In transfer learning in CNN, we utilize the early and central layers, while only retraining the latter layers. The model leverages labeled data from its original training task. In our example, if a model originally trained to identify backpacks in images now needs to detect sunglasses, it uses its prior learning.

So the idea is based on the fact that the few first layers of CNN only extracts low-level and **frequent** features, these are mostly seen in all different types of images and classes and are not specific, while the last few layers of CNN are mostly **task-specific** resulting in only training the last few layers of the pre-trained model **fine-tunes** the model for the specific task and considerably increases the accuracy.