Question 2: Classification: Convolutional Neural Networks (60 points)

2.1: Design and Implementation Choices of your Model (25 points)

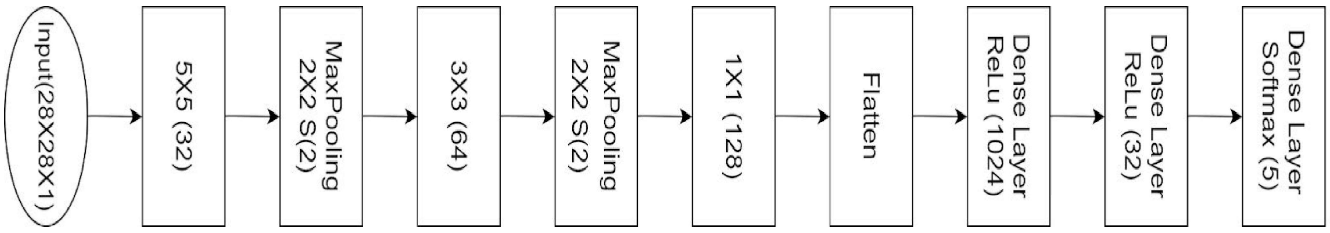
• Use some CNN-based approach to solve the classification problem. You can explore any architecture of the network you like.

Implemented model features a convolutional neural network architecture. The architecture presented to solve classification problems is inspired from popular CNN architectures Lenet5 [1], Alexnet [2] and VGG [3] network architectures. The architecture is designed with basic patterns explained in above mentioned traditional CNN architectures. The model Implementation and architecture is described below:

Model Implementation and Architecture:

1. **Data pre-processing**

   The model is implemented on the FashionMNIST dataset as given in kaggle ECE657A Competition. The complete training dataset is splitted in the two parts i.e. Training dataset (90%) and validation (10%) dataset. Before feeding the data to the model, all the features (784 pixel values) are brought to the same scale i.e. between 0 and 1 (scaling is performed for better convergence). The features are scaled by applying min-max transformation(i.e. Each pixel value is divided by 255). Each sample image is reshaped to the 28x28x1 tensor before being fed to the CNN model.
2. **Model architecture**



   Architecture implemented for this classification problem is based on the LeNet, AlexNet and VGG network models. The network architecture comprises many Convolutional layers with variation in filter size and channel depths. The idea behind design of model architecture is to reduce the rectangular field size with gradual increase in channel depths resulting in the deantangled features extraction. The selected features are passed through the dense layers of different neuron sizes. The detailed model is as follows.

   1. The input layer takes the image of 28x28x1 size.
   2. The first convolution layer consists of a convolutional 2D layer of kernel size 5x5 and kernel depth 32 with the same padding. This layer is followed by the Max pooling layer of the window size 2x2 and stride 2. The output of the first convolutional-pooling layer stack is of size 14x14x32.
   3. Output of the above layers is passed through another convolutional-pooling layer combination in order to increase the channel depth and reduce the output size. The convolutional layer has a filter size 5x5 and 64 filters with same padding which is followed by the Max Pooling layer of 2x2 window size and strides 2. The size of the image after this layer is 7x7x64.
   4. To further increase and extract the disentangled features of the image, output of the above pooling layer is passed through the convolutional layer of filter size 1x1 and 128 filters resulting in increase of image channel numbers to 128. To get the regularising effect and avoid overfitting of the model, drop out of 0.2 is used after this layer.
   5. Since the pixel size of the input image is small (i.e. 28x28), it is assumed that unique features of sample image are extracted as the model proceeds till the above mentioned layer

and hence dense layers are used to process these features further. To convert these disentangled into dense neural layers the output of convolutional layer 1x1 (128 features) is flattened.

6. The flatten output is being fed to the two dense layers of different numbers of neurons. The first layer consists of 1024 numbers of neurons followed by the layer with 32 neurons . The activation function used in these layers is a rectified linear unit (ReLu) to avoid saturation of gradients.
7. The last dense layer of the model is using a softmax activation function to predict the classes based on the probability.

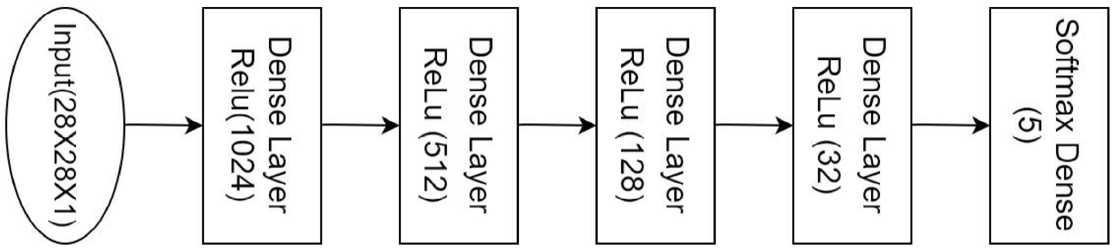
3. **Optimizers**
For optimizing the cost function of the neural network one of the adaptive learning rate models that is Adaptive Moment Estimation (Adam) [7] is used. Adam overcomes the decaying learning rate problem of optimizor Adagrad [7]. Moreover, it also solves the drawback of slow convergence and High variance of internal parameters observed in the gradient descent optimization algorithm. Adam being a state of the art optimization algorithm, it solves various drawbacks of other optimization algorithms and hence is implemented in this model for robust convergence of gradients.
4. **Regularisor**
To reduce the model overfitting, dropout technique is used as a regulator. It prevents overfitting and provides a way of approximately combining exponentially many different neural network architectures efficiently [8]. The term “dropout” refers to dropping out units (hidden and visible) in a neural network. By dropping a unit out, we mean temporarily removing it from the network, along with all its incoming and outgoing connections. The choice of which units to drop is random.
5. **Activation Functions**
Activation function for convolutional and dense layers is kept as a rectified linear unit function(ReLu), and for the output layer softmax function as activation function is used. ReLu activation is used to avoid the gradient saturation problem whereas softmax is used for selecting the parameter associated with maximum probability value.
6. **Weight initialization**
The Glorot_uniform initializer is used for initializing weights for neural layers. This initialization is used to prevent redundant learning in the neural network.
7. **Libraries Used**
For implementation of this model, tf.keras library is used which is TensorFlow's high-level API for building and training deep learning models.

• You should also consider exploring other ML methods and Deep Neural Network variants to solve the problem, cite sources you used: libraries, as well as papers or blogs. (eg. can the use of Resnet learn a better classier than a simple CNN?)

Other Explored Models:

1. Dense Neural networks:

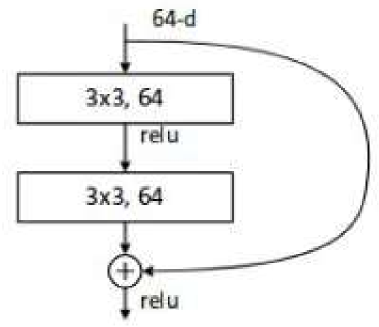
A densely connected network of Artificial neural layers (similar to basic deep neural network, refer Appendix A) is used here for the image classification problem. The architecture for densely connected network used to categorize Fashion MNIST images is shown below:



The above shown Resnet network based CNN model features a stack of convolutional and max pooling layers, with a total of 5 layers in the model.

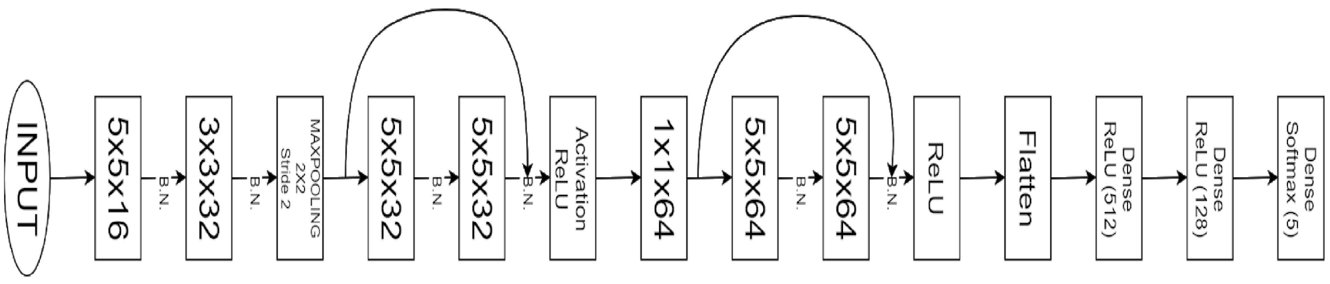
2. Resnets:

Skip connection based neural networks, introduced in [4] are also explored to improve accuracy of dense connected networks. The models were introduced to prevent the problem of vanishing gradients in deep neural networks. A skip connection based residual block looks like the one shown in Figure below :



A residual neural network based model, may feature one or more such residual blocks. These models enable us to train very deep neural network architectures, without facing problems of degradation of gradients.

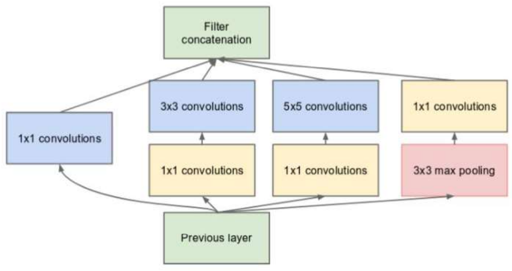
The Residual network architecture used to classify Fashion MNIST data for this assignment is shown below:



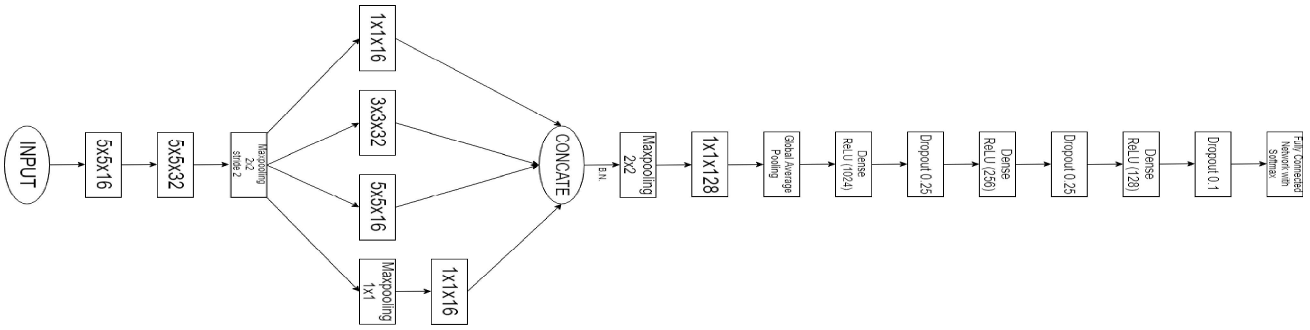
The above shown Resnet network based CNN model features a total of 11 layers with stacks of convolutional and max pooling layers, two skip connections, flattened layers and dense layers.

3. Inception network:

Models inspired from sparsely connected convolutional networks introduced in Googlenet [5] have also been explored for solving this classification problem. These model feature inception blocks; which are formed by using a combination multiple convolution filters, 1x1 networks and pooling layers. A typical Inception block looks like the one shown in figure [6] below.



The Inception network inspired architecture used to classify Fashion MNIST data for this assignment is shown below:

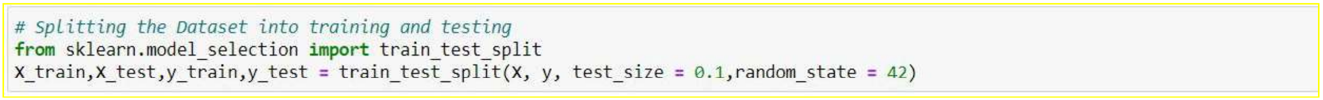


The above shown inception network based CNN model features 9 neural layers with a stack of convolutional and max pooling layers,one inception block, global average pooling layer and dense layers.

2.3: Implementation of your Design Choices (10 points)

Show some of the important code blocks to implement your model. We will also consult your full code on LEARN, so this is your chance to guide us to understand your code and how you achieved your result.

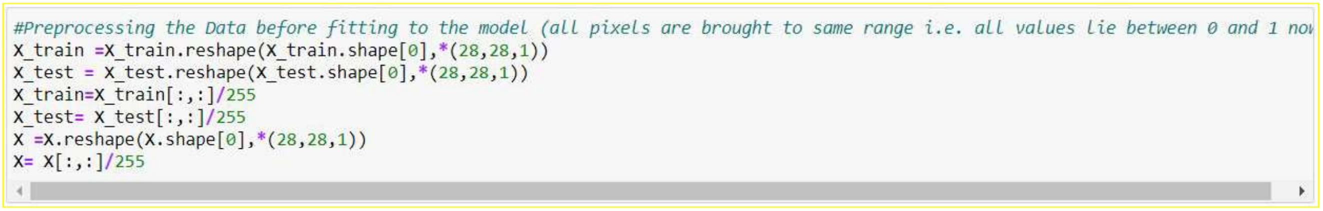
1. **Splitting the data**



```
# Splitting the Dataset into training and testing
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X, y, test_size = 0.1,random_state = 42)
```

The complete training dataset is splitted in the two parts i.e. Training dataset (90%) and validation(10%) dataset. The performance on validation set is used to compare results of various algorithms and architecture used.

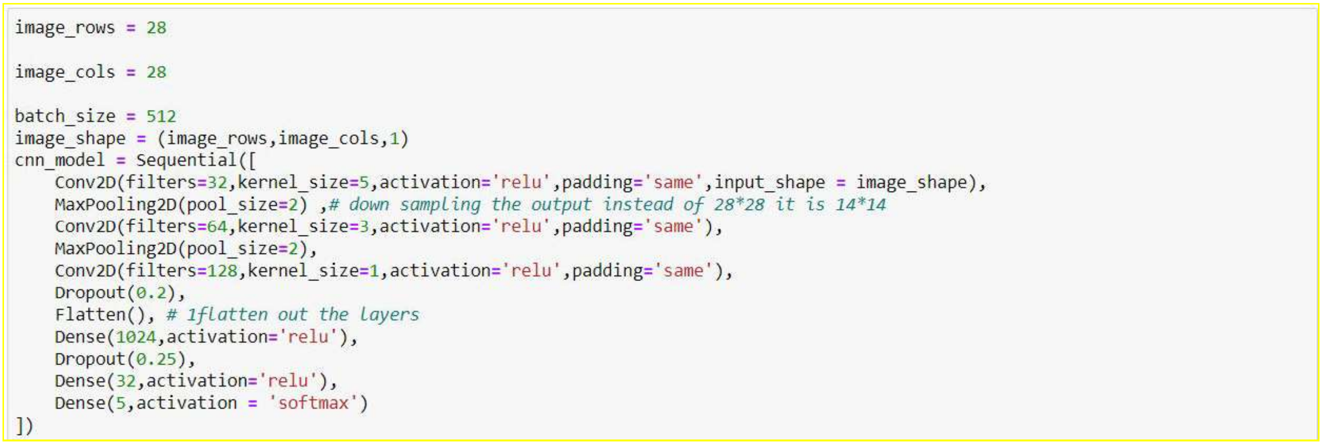
2. **Pre-processing of the data**



```
#Preprocessing the Data before fitting to the model (all pixels are brought to same range i.e. all values lie between 0 and 1 now
X_train =X_train.reshape(X_train.shape[0],*(28,28,1))
X_test = X_test.reshape(X_test.shape[0],*(28,28,1))
X_train=X_train[:,:]/255
X_test= X_test[:,:]/255
X =X.reshape(X.shape[0],*(28,28,1))
X= X[:,:]/255
```

Before feeding the data to the model, all the features (784 pixel values) are brought to the same scale i.e. between 0 and 1 (scaling is performed for better convergence). The features are scaled by dividing with maximum value possible i.e. 255 (min-max transformation). Each sample image pixel is reshaped to the 28x28x1 tensor.

3. **Architecture of the model**

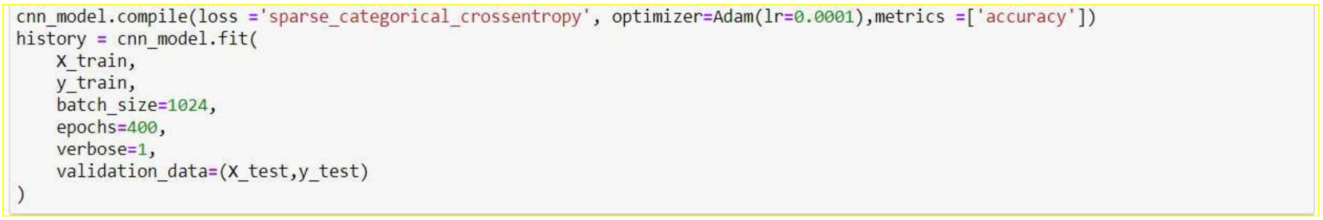


```
image_rows = 28

image_cols = 28

batch_size = 512
image_shape = (image_rows,image_cols,1)
cnn_model = Sequential([
    Conv2D(filters=32,kernel_size=5,activation='relu',padding='same',input_shape = image_shape),
    MaxPooling2D(pool_size=2) ,# down sampling the output instead of 28*28 it is 14*14
    Conv2D(filters=64,kernel_size=3,activation='relu',padding='same'),
    MaxPooling2D(pool_size=2),
    Conv2D(filters=128,kernel_size=1,activation='relu',padding='same'),
    Dropout(0.2),
    Flatten(), # 1flatten out the layers
    Dense(1024,activation='relu'),
    Dropout(0.25),
    Dense(32,activation='relu'),
    Dense(5,activation = 'softmax')
])
```

Above snippet shows the architecture of the sequential model made up of the different convolutional and dense layers.

1. Initial convolutional layer has kernel size 5 and depth 32 with the same padding. The reason behind the large window size is to find the relation of each pixel with its neighbour pixels. The activation used in all the convolutional layers is a rectified linear unit(ReLu). This layer is followed by the max pooling layer of size (2,2) and stride 2 resulting in the output size of 14x14x32.
2. To extract disentangled features, a second convolutional layer of 3x3 window and filter depth 64 is added with the same padding followed by the max pooling layer of pool size (2,2) and stride 2.
3. To increase the channel depth further, a simple convolutional layer of 1x1 filter and 128 depth added to the model with the same padding. In order to avoid overfitting and get the regulation effect, drop out of 0.2 is used.
4. It is assumed that the features required to classify the data are extracted at this stage and hence The extracted features are flattened before passing it to the dense layers for classification.
5. The flatten output is passed through the dense layer of 1024 neurons with ReLu activation function and dropout of 0.25.
6. Second dense layer consist 32 number of neurons with ReLu activation followed by the dense layer with softmax activation of 5 neurons is used as output layer.

4. **Model compilation**



```
cnn_model.compile(loss ='sparse_categorical_crossentropy', optimizer=Adam(lr=0.0001),metrics =['accuracy'])
history = cnn_model.fit(
    X_train,
    y_train,
    batch_size=1024,
    epochs=400,
    verbose=1,
    validation_data=(X_test,y_test)
)
```

Model is compiled using the sparse_categorical_crossentroty loss function and is optimized using the Adam optimiser with 0.0001 learning rate. For model evaluation, accuray metric is used.

2.4: Kaggle Competition Score (5 points)

Report the highest score your submissions received on Kaggle. If you have any explanations for varying performance by different teams explain it here.

91.84 % Accuracy is achieved on the leaderboard implementing the CNN based model (inspired by Lenet, Alexnet and VGG net). The model is trained on the complete training data set provided to achieve this accuracy.

There are various parameters that can be changed to improve accuracy scores. Different teams might have used different optimizers like SGD, ADM, ADAbound (available in pytorch) and some might have designed different architectures like Resnet, Googlenet to achieve different results. Since, these networks along with different parameters might get stuck on local optimals, it becomes difficult to find the best combination of tuned parameters with problem specific model architecture. Therefore, the variation in different performances can be attributed to choice of hyperparameters, random_states (though would not have a huge impact) and choice of CNN architecture.

2.5: Results Analysis (20 points)

• Runtime performance for training and testing.

Time taken to run one epoch is approximately 1 seconds and time taken to test the validation data is approximately 0.2 seconds. Since a total of 400 epochs are performed, so total training time is approximately 400 seconds. The training time is dependent on RAM and CPU core availability and varies dynamically as system configuration is made available by google collaborator at the given time.
(All calculations are done at same system configurations made available by google collaboratory).

• Comparison of the different algorithms and parameters you tried.

Following table shows performance of different neural network based (best performing models are shown) approaches with epochs 400, and a batch size of 1024.

Model Architecture Model Performance	**Dense ANN**	**Traditional CNN**	**Resnet based Models**	**Inception based Models**
Training Time per epoch	0.47 seconds	1 second(appx)	5 seconds(appx)	4 Seconds(appx)
No of Trainable Parameters	1,398,597	6,484,330	6,753,669	325,426
Layers	5	7	11	9
Training Loss	0.0411	0.0327	0.012	0.0153
Validation Loss	0.8803	0.2995	0.6031	0.5963
Training Accuracy	0.9852	9891	0.999	0.9952
Validation Accuracy	0.8619	0.9137	0.897	0.9027

Following observations can be made from above table:

1. CNN based approaches such as traditional CNN, Resnets and Inception networks perform(on training as well as validation set) comparatively better on image dataset than Densely connected neural network architectures, but take more time to train.
2. For the approximately same number of parameters the training time of Resnet based models is significantly more than that of Traditional CNN models.
3. Inception models might take longer time to train compared to similar traditional CNN models even with less number of parameters. On the other hand, performance of Inception networks on training and validation data is very close to performance of traditional CNN architectures with one twentieth of parameters being used.
4. To achieve accuracy scores similar to traditional CNN models, Resnets based models tend to have a deeper layered network.
5. Training and validation losses are negatively related with training and validation accuracies respectively.

Following table shows performance of different activation functions on the traditional CNN model described above with Adam optimizer (learning rate =0.001) approaches with epochs 400, and a batch size of 1024:

Activation Function	Training Cost	Validation Cost	Training accuracy	Validation Accuracy
Sigmoid	0.2427	0.2518	0.905	0.9013
ReLu	0.0327	0.2995	0.9891	0.9137
Tanh	0.063	0.2928	0.978	0.9023

Following Observations can be made from the Table :

1. Relu activation provides best training and validation accuracy among Sigmoid, Relu and Tanh for this model at a fixed learning rate of 0.001.
2. Sigmoid activation function has a tendency to saturate and Tanh activation function performed better on this CNN model compared to Sigmoid, on the above mentioned model architecture and learning rate.

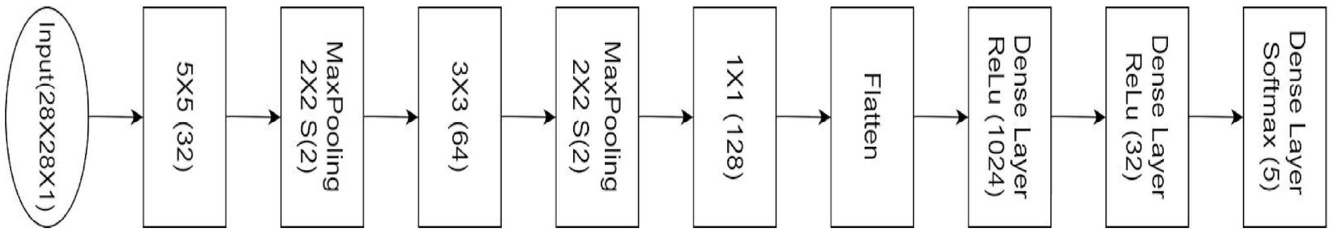
Following table shows performance of different optimizers on the traditional CNN based model described above with activation function ReLu approaches with learning rate of 0.0001, epochs = 400 and a batch size of 1024.

Optimization	Training Cost	Validation Cost	Training accuracy	Validation Accuracy
RMSprop	0.0231	1.0896	0.9932	0.8625
Adam	0.0327	0.2995	0.9891	0.9137
SGD	0.2679	0.2622	0.8926	0.8972

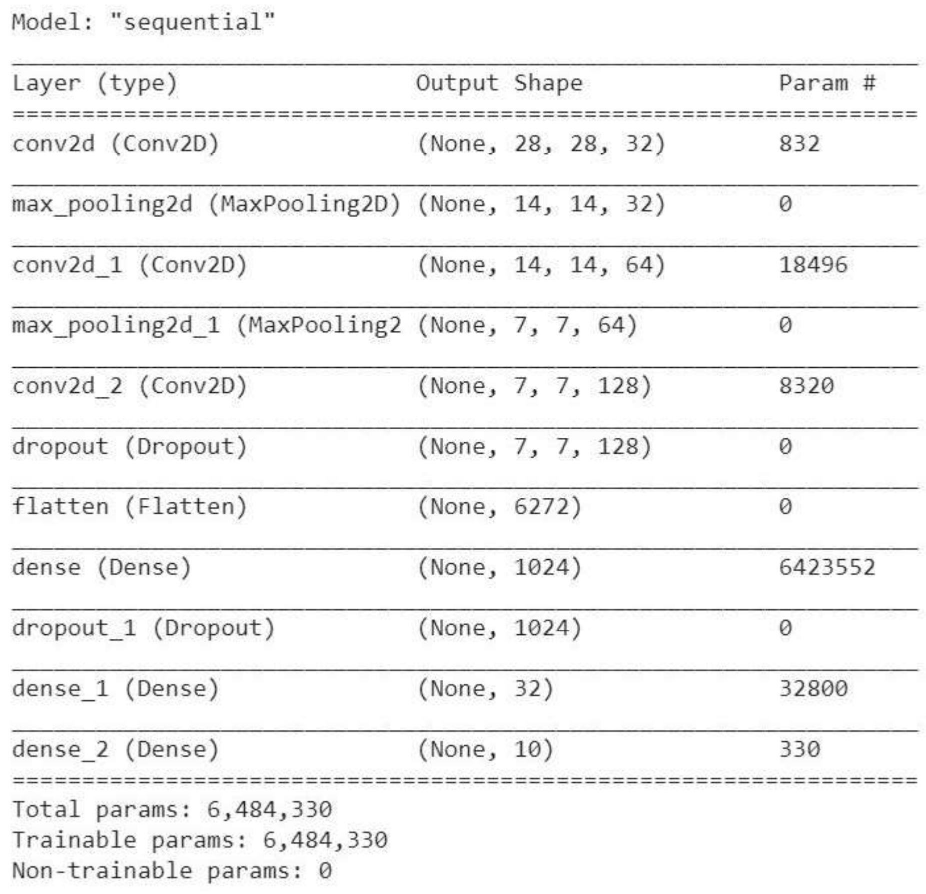
Following observations can be made from the above table:

1. Minimization of loss function with RMSprop optimizer overfits the model more compared to Adam and RMSprop optimization algorithms.
2. SGD optimizer may take more epochs to achieve similar training and validation accuracy as compared to RMSprop and Adam optimizer.

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



The architecture of our model is explained in question 2.1. The loss function used while compiling this model is “sparse_categorical_crossentropy” as we have multi class numerical labels alternatively we could have used the “categorical_crossentropy” if we had hot encoded the labels. Moreover, this loss function is minimized using the Adaptive Moment Estimation (Adam). Adam is prefered over Adagrad and SGD as it overcomes the decaying learning rate problem of optimizer Adagrad and drawback of slow convergence and High variance of SGD optimisation algorithm. To get the regularising effect and avoid overfitting of the model, drop of 0.2 and 0.25 is added in the convolutional layer of 1x1 (128) and dense layer of 1024 neurons respectively. The number of trainable parameters in all the layers are shown in the below image,



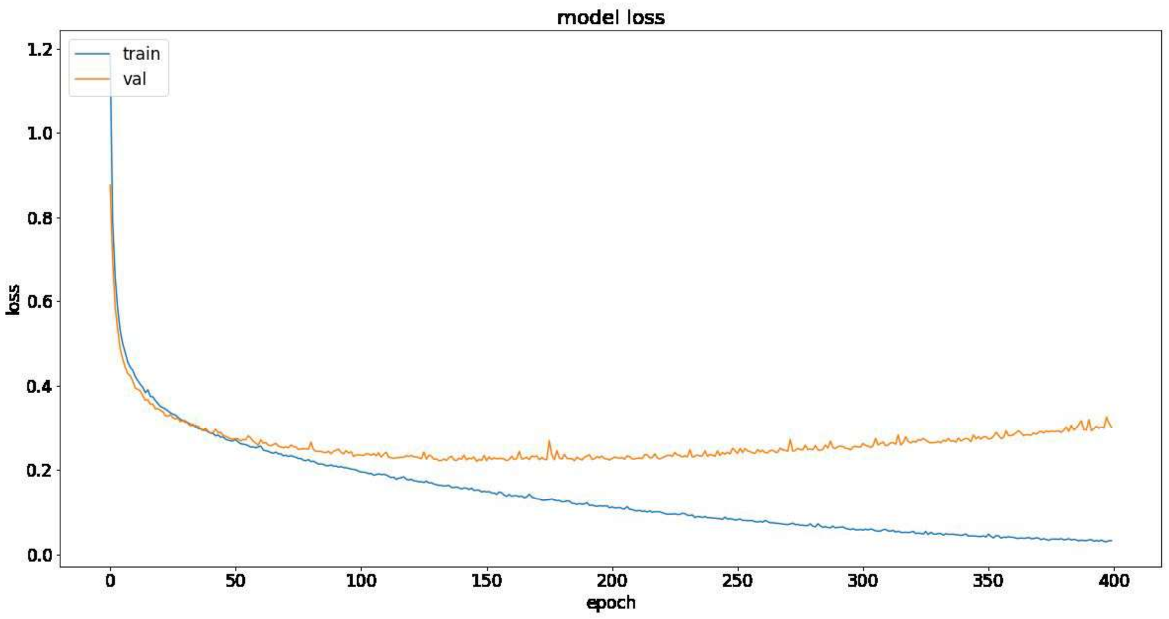
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 32)        832
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 14, 14, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 7, 7, 128)         8320
_________________________________________________________________
dropout (Dropout)            (None, 7, 7, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 6272)              0
_________________________________________________________________
dense (Dense)                (None, 1024)              6423552
_________________________________________________________________
dropout_1 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 32)                32800
_________________________________________________________________
dense_2 (Dense)              (None, 10)                330
=================================================================
Total params: 6,484,330
Trainable params: 6,484,330
Non-trainable params: 0
```

Total number of trainable parameters from the above image are 6.4 million. The maximum number of trainable parameters introduced in the model are by the fully connected layers.

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

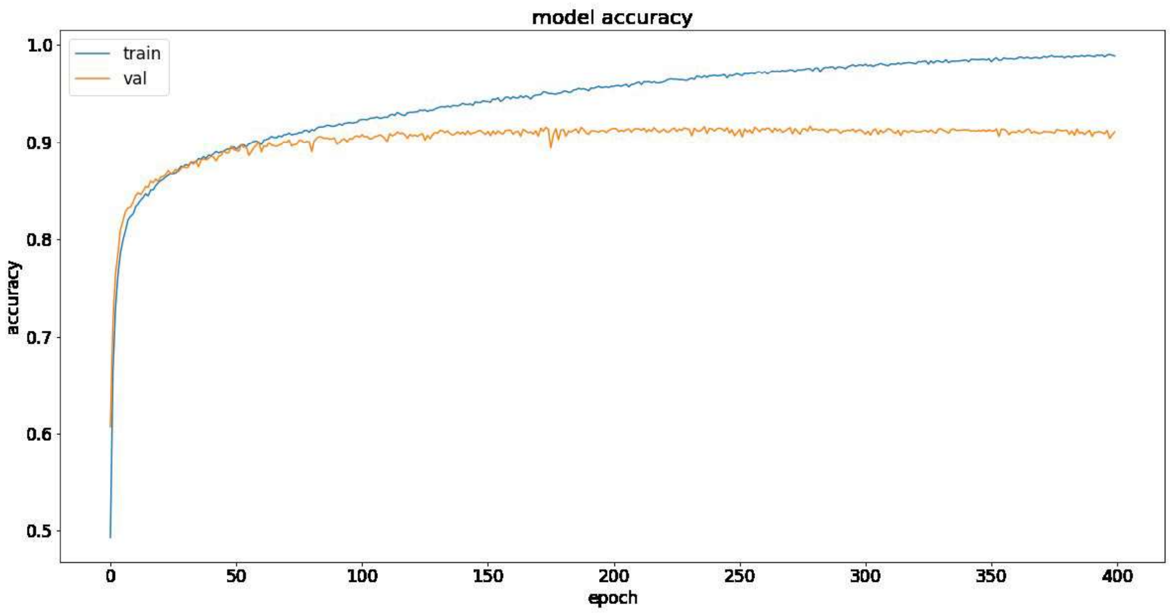
Performance of the model can be explained using the plots of Loss Vs. Epoch and Accuracy Vs. Epoch for both the training and testing (validation) dataset.

Epoch Vs. Loss



Above plot shows the values of the loss function of both the training and validation dataset for each number of the epoch.The model is run for 400 number of epochs. It can be seen from the graph that till the first 50 epochs, the loss value of both the training and validation data set decrease rapidly. With further increase in number epochs, training loss still follows the decreasing trend. On the other side, the loss value of the validation set gets constant between epoch numbers 50 and 200. After that it follows a mild increasing trend.

Epoch Vs. Accuracy



Above plot shows the values of the accuracy of both the training and validation dataset for each number of the epoch.The model is run for 400 number of epochs. It can be seen from the graph that till 50 epochs, the loss value of both the training and validation data set increases rapidly. With further increase in number epochs, training accuracy follows the increasing trend till the last number of epochs. On the other side, the accuracy of the validation test is almost constant after 50 number of epochs. This shows mild overfitting of the model. However, the model seems decent as it gets a validation accuracy of approximately 91.36%.

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

The model can be evaluated on the different accuracy matrices such as accuracy, F1-score, ROC AUC etc. Values for different accuracy matrices are shown in the table below.

Accuracy	91.36%
Precision	91.36%`
Recall	91.36%
F1-Score	91.36%
ROC AUC	0.9923

To understand model performance on validation better, various metrics such as Accuracy, precision, recall, F1-Score and Area under the AUC are used as well. The precision value tells about the sensitivity of the model whereas recall value explains true positive rate. F1-score shows the balance between precision and recall.Similar high values for precision (91.36%) and recall (91.36%) on validation data confirms the model's excellent ability to distinguish between different classes (true positives, false positive and false negative values are taken into consideration to calculate precision, recall and F1 score) classification. High value of F1 score shows the model's balance between precision and recall ability. Furthermore high value (.9923, i.e. close to 1) of area under the ROC (Receiver Operating Curve) shows high true positive rate in the model's ability to classify images into respective categories.

References:

1. Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278-2324, Nov. 1998
2. Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." In *Advances in neural information processing systems*, pp. 1097-1105. 2012.
3. Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." *arXiv preprint arXiv:1409.1556* (2014).
4. He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770-778. 2016.
5. Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going deeper with convolutions." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 1-9. 2015.
6. https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202, Accessed on [April 20,2020]
7. [https://towardsdatascience.com/types-of-optimization-algorithms-used-in-neural-networks-and-ways-to-optimize-gradient-95ae5d39529f].
8. N. Srivastava, G. Hinton, A. Krizhevsky, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," 2014.