Convolutional Neural Network vs Fully Connected Neural Network for Image Recognition

Student no: 18821327
School: University of Brighton, School of Computing, Engineering and Mathematics

I confirm that I have a Learning Support Plan for 'spelling and grammar and extension' as recommended by the Disability and Dyslexia Team, and agreed by the School. I understand that the deadline for my assessment has been adjusted (as per the required School protocol) and that this, and my spelling and grammar, should be taken into consideration when my assessment is marked/ graded.

Table of Contents

Introduction

'One of the most challenging multi-classes classification problems is fashion classification in which labels that characterize the clothes type are assigned to the images. The difficulty of this multi-classes fashion classification problem is due to the richness of the clothes properties and the high depth of clothes categorization as well. This complicated depth makes different labels/classes to have similar features.' (Kayed, Anter and Mohamed, 2020) Because of this I have chosen to use the fashion mnist dataset to compare the accuracy, precision and loss of 2 deep neural networks (DNNs), a convolutional neural network (CNN) and a fully connected neural network (FCNN). For which, the best performing model of the two will have their accuracy compared against that of other models in literature which were also tested on the fashion mnist dataset.

For humans image recognition is a trivial task, 'this is because our brains have been trained unconsiously with the same set of images that has resulted in the development of capabilities to differentiate between things effortlessly.' (Gupta 2018) However, a 'computer views visuals as an array of numerical values and looks for patterns in the digital image ... to recognise and distinguish key features of the image.' (Gupta, 2018)

Image recognition algorithms can be seen in everywhere in our lives today. In smartphones, government and banking apps they all employ a type of image recognition software. For example; most smartphones now employ facial recognition software, for verify your id on government sites you are able to scan your id/passport with your phone, and for banks you are able scan cheques from your home to deposit into your account. Because of this we need algorithms that are able to perform with a high degree of accuracy and precision, otherwise you could get locked out your phone, your id verification could fail or you could recieve the wrong amount of money but not at the fault of the user.

In recent years the field of computer vision has grown massively due to the demand for autonomous and semi-autonomous vehicles and drones. In this field lies a sub-field called image recognition, here CNNs have become the standard DNN for solving these types of problems. This is because in CNNs 'feature extraction is figure out by itself and these models tend to perform well with huge amount of samples.' (Greeshma and Sreekumar, 2019)

Loss, optimiser and performance mertics

The loss function tells the optimizer if it is changing the weights and biases in the correct direction, for example after the optimizer changes the weights and biases and recieves a higher loss the optimizer is moving in the wrong direction. Categorical Crossentropy is the chosen loss function. Categorical Crossentropy tells you the difference between 2 probability distributions.

The optimiser is used to update the weights and biases of the nodes in the network based on the value of the loss function and the learning rate. 'Adam, an adaptive learning rate method, will compute individual adaptive learning rates for each parameter based on the average of the mean (first moment) and the average of the uncentered variance (second moment). Each of these averages (moments) will have a decay rate controlled by parameters, beta 1, beta 2, respectively applied to them during the training phase.' (Brownlee, 2021)

Categorical accuracy and precision are the chosen performance metrics. Accuracy is used to described how the model performs across all classes. It is calculated as the ratio between the number of correct predictions to the total number of predictions. Precision tells us how accurate the model is at predicting a sample as positive. It is calculated as the ratio between the number of positive samples correctly classififed to the total number of sampled classified as positive.

What is a Deep Neural Network

Let's start with what is a neural network, 'artificial neural networks (ANNs) are comprised of node layers, containing an input layer, one or more hidden layers, and an output layer. Each node, or artificial neuron, connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network.' (IBM Cloud Education, 2020)

A deep neural network, is a neural network that consist of 2 or more hidden layers. A hidden layer is any layer between the input and output layer. They are considered hidden layers because they are not directly obserable from the input and output layers.

Imports

In [1]:

```
import datetime
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from tensorflow.keras import Input, Sequential
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.layers import Conv2D, Dense, Flatten, MaxPool2D, Dropou
from tensorflow.keras.losses import CategoricalCrossentropy
from tensorflow.keras.metrics import CategoricalAccuracy, Precision
from tensorflow.keras.models import load_model
from tensorflow.keras.optimizers import Adam, SGD
from tensorflow.keras.utils import to_categorical
import time
```

Load training and testing data

In [2]:

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

The dataset

The fashion mnist dataset contains a total of 70000 samples: 60000 training and 10000 testing samples. Each sample contains 784 features and 1 label, where the features are a value from 0 to 255 detailing the lightness of a pixel, with the value closest to 0 showing a black pixel and the value closest to 255 a white pixel. The 10 labels are:

- 0 - T-shirt/top
- 1 - Trousers
- 2 - Pullover
- 3 - Dress
- 4 - Coat
- 5 - Sandal
- 6 - Shirt

- 7 - Sneaker
- 8 - Bag
- 9 - Ankle boot

In the training samples there are 6000 of each fashion clothing item and 1000 of each in the testing samples.

Visualising the dataset

In [3]:

```
NUM_CLASSES = 10
data = {
    'Training Samples': x_train.shape[0],
    'Testing Samples': x_test.shape[0],
    'Features': x_train.shape[1] * x_train.shape[2],
    'Labels': NUM_CLASSES
}
df = pd.DataFrame(data, index=['Amount'])
df
```

Out[3]:

	Training Samples	Testing Samples	Features	Labels
Amount	60000	10000	784	10

In [4]:

```
# Find the amount of times each value occurs in the dataset
image_names = ['T-shirt/top', 'Trousers', 'Pullover', 'Dress', 'Coat', 'Sanda
y_explore = (y_train, y_test)
frequencies = []
for _ in np.arange(2):
    (unique, counts) = np.unique(y_explore[_], return_counts=True)
    frequencies.append(np.asarray((unique, counts)).T[:, 1])

# Display in a dataframe
indexs = ['Training', 'Testing']
df = pd.DataFrame(frequencies, index=indexs, columns=image_names)
df
```

Out[4]:

	T-shirt/top	Trousers	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle boot
Training	6000	6000	6000	6000	6000	6000	6000	6000	6000	6000
Testing	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000

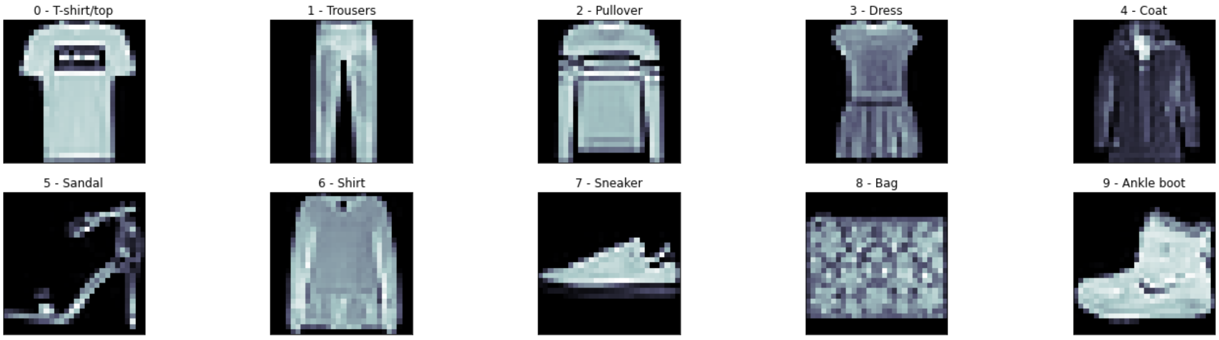
In [5]:

```
found = [x for x in range(10)]
label_values = []
for i in range(24):
    label = y_train[i]
    values = x_train[i]
    if label in found:
        label_values.append((label, values))
        found.remove(label)

# sort items based on value of label
label_values.sort(key=lambda label: label[0])
```

Below you can see 1 example from each of the 10 label sets.

In [6]:

```
fig = plt.figure(figsize=(20, 5), tight_layout=True)
for idx in np.arange(10):
    ax = fig.add_subplot(2, 5, idx+1, xticks=[], yticks=[])
    ax.imshow(np.squeeze(label_values[idx][1]), cmap='bone')
    ax.set_title(f'{label_values[idx][0]} - {image_names[idx]}')
```



Preprocessing the dataset

At the moment, the data is not ready to be fed to the networks.

The x data, also known as the features, needs to be converted from its 0 - 255 scale to 0 - 1 scale. This is done by converting the integer numbers to decimal numbers and then dividing each number in the x data by 255.0. Since the each row of the x data is currently in a 28 x 28 2 dimensional array, the dimensions need to be expanded so that they have a depth of 1, this causes the values to be reshaped into 1 dimensional array, so they can be fed as input to the networks.

The y data, also known as the labels, needs to have its values converted from integers representing the item, into binary numbers representing the item. For example, before conversion each value in the y data set would contain one integer representing the item, 1 for Trousers, after conversion each value contains an array of binary numbers, the index of the 1 in the array points to the item that the feature values correspond to, [0, 1, 0, 0, 0, 0, 0, 0, 0, 0] for Trousers. This is done so that later on the network can asign a probability to each index in the array, where the probability closest to 1 indicates the networks prediction.

In [7]:

```
print('-------Before processing-------')
print(f'x_train shape: {x_train.shape}')
print(f'x_test shape: {x_test.shape}\n\n')


print(f'y_train shape: {y_train.shape}')
print(f'y_test shape: {y_test.shape}\n\n')

x_train = x_train.astype('float32') / 255.0
x_train = np.expand_dims(x_train, -1)

x_test = x_test.astype('float32') / 255.0
x_test = np.expand_dims(x_test, -1)

y_train = to_categorical(y_train, NUM_CLASSES)
y_test = to_categorical(y_test, NUM_CLASSES)


print('-------After processing-------')
print(f'x_train shape: {x_train.shape}')
print(f'x_test shape: {x_test.shape}\n\n')
```

```
print(f'y_train shape: {y_train.shape}')
print(f'y_test shape: {y_test.shape}')
```

```
-------Before processing-------
x_train shape: (60000, 28, 28)
x_test shape: (10000, 28, 28)


y_train shape: (60000,)
y_test shape: (10000,)


-------After processing-------
x_train shape: (60000, 28, 28, 1)
x_test shape: (10000, 28, 28, 1)


y_train shape: (60000, 10)
y_test shape: (10000, 10)
```

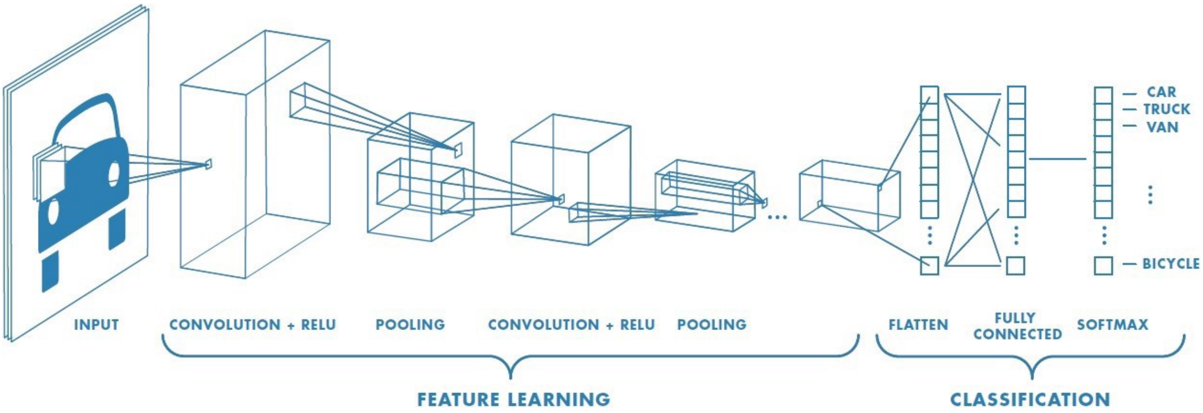
In [8]:

```
# model parameters
INPUT_SHAPE = (28, 28, 1)
KERNEL_SIZE = (5, 5)
POOL_SIZE = (2, 2)
PADDING = 'same'
CONV_UNITS = 32
FULLY_UNITS = 64
ACTIV = ['relu', 'softmax']
DROPOUT = 0.3

# training parameters
BATCH_SIZE = 300
EPOCHS = 15
LOSS = CategoricalCrossentropy()
OPTIMIZER = Adam(learning_rate=1e-3)
EARLYSTOPPING = EarlyStopping(monitor='val_loss', mode='min', patience=4)
METRICS = [CategoricalAccuracy(), Precision()]
```

What is a Convolutional Neural Network

Simply put, a convolutional neural network is a deep learning algorithm which can take in an input image, assign learnable weights and biases to various aspects in the image and be able to differentiate one from the other. However, to expand upon this, I found the following explanation of how the CNN solves image recogntion problems:

'The inputs of CNN are not fed with the complete numerical values of the image. Instead the complete image is divided into a number of small sets with each set itself acting as an image. A small size of filter divides the complete image into small sections. Each set of neurons is connected to a small section of the image. These images are then treated similar to the regular neural network process. The computer collects patterns with respect to the image and the results are saved in the matrix format. This process repeats until the complete image in bits size is shared with the system. The result is a large matrix, representing different patterns the system has captured from the input image. This matrix is again downsampled (reduced in size) with a method known as max pooling. It extracts the maximum values from each sub matrix and results in a matrix of much smaller size. These values are representative of the pattern in the image. This matrix formed is supplied to the neural networks as the input and the output determines the probability of the classes in an image.' (Gupta, 2018)



(Saha, 2018)

CNN architecture

Below creates CNN models with different architectures. Where after each model is trained the next model has another Convolutional and MaxPool layer pair added after the first once there are 3 pairs another fully connected layer is added. With the first architecture having 1 Convolutional and MaxPool layer pair and 1 Fully Connected layer. The last having 3 Convolutional and MaxPool layer pairs and 2 Fully Connected layers. This was done so that I could find the optimal architecture for my simple CNN model.

Each layer in a network has an activation function where an activation function take as input the previous stage's output and apply a mathematical function to that input, in the case of 'relu' if the input is less than 0 it is changed to 0 and any number equal to or greater than 0 is left unchanged, for 'softmax' the values are scaled down to probabilies, where the value closest to 1 presents the chosen output.

The Convolutional and Fully connected layers both use the 'relu' activation function. The output layer has the 'softmax' activation function.

In [9]:

```
def cnn_arch(n, m):
    model = Sequential()
    # input layer
    model.add(Input(INPUT_SHAPE))
    # convolutional & pooling layers
    for _ in np.arange(n):
        model.add(Conv2D(CONV_UNITS, kernel_size=KERNEL_SIZE, activation=ACTI
        model.add(MaxPool2D(pool_size=POOL_SIZE))
    # flatten layer
    model.add(Flatten())
    # fully connected layers
    for _ in np.arange(m):
        model.add(Dense(FULLY_UNITS, activation=ACTIV[0]))
    # output layer
    model.add(Dense(NUM_CLASSES, activation=ACTIV[1]))
    return model
```

CNN training

Here each model is compiled with a loss function, an optimser and performance metris. Then the models are fitted with the training data, where they are trained over 15 epochs.

For each epoch the training data is split (80/20) at differnt places in the data for each epoch. The model is trained on the 80% and then validated on the 20%. When the model is training the following performance metrics are being recored; 'Categorical Crossentropy', 'Categorical Accuracy' and 'Precision'.

When the model is training on the 80% the model is fed inputs and the values are passed through the nodes of the network to the output layer where the model predicts an output. Now the loss function, categorical accuray and precision are calculated using the predicted output and back propagation is used to tune weights of the nodes in the network. After which, the model, with its tuned weights is validated on the 20% to see how it will perform on unseen data later on during the testing phase.

Early stopping is used during the training phase to stop training a model based upon the parameters passed in. The parameters I will pass in, will be used to detect when the models val_loss has stopped decreases with a patience of 3. Meaning after 3 epochs of the val_loss increasing the model will stop being trained. This is a method of preventing overfitting, because it stops the models training early when it realises the model is not generalising well.

In [10]:

```
def training(arch, name):
    historys = []
    count = 1
    if name == 'cnn':
        conv = 1
        hidd = 1
        while hidd < 3:
            model = arch(conv, hidd)
            model.summary()
            model.compile(loss=LOSS, optimizer=OPTIMIZER, metrics=METRICS)
            history = model.fit(x_train, y_train, BATCH_SIZE, EPOCHS, validat
            historys.append(history)
            model.save(f'{name}/{name}.{count}')
            count += 1
            conv += 1
            if conv == 4:
                hidd += 1
                conv = 1
    else:
        while count < 5:
            model = arch(count)
            model.summary()
            model.compile(loss=LOSS, optimizer=OPTIMIZER, metrics=METRICS)
            history = model.fit(x_train, y_train, BATCH_SIZE, EPOCHS, validat
            historys.append(history)
            model.save(f'{name}/{name}.{count}')
            count += 1

    return count-1, historys
```

In [11]:

```
cnn_count, cnn_history = training(cnn_arch, 'cnn')
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 32)        832
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 32)        0
```

```
_________________________________________________________________
flatten (Flatten)            (None, 6272)              0
_________________________________________________________________
dense (Dense)                (None, 64)                401472
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 402,954
Trainable params: 402,954
Non-trainable params: 0
_________________________________________________________________
Epoch 1/15
160/160 [==============================] - 16s 93ms/step - loss: 0.8637 - cate
gorical_accuracy: 0.7089 - precision: 0.8338 - val_loss: 0.4185 - val_categori
cal_accuracy: 0.8513 - val_precision: 0.8854
Epoch 2/15
160/160 [==============================] - 15s 96ms/step - loss: 0.3763 - cate
gorical_accuracy: 0.8672 - precision: 0.8968 - val_loss: 0.3547 - val_categori
cal_accuracy: 0.8752 - val_precision: 0.9019
Epoch 3/15
160/160 [==============================] - 15s 95ms/step - loss: 0.3226 - cate
gorical_accuracy: 0.8882 - precision: 0.9096 - val_loss: 0.3235 - val_categori
cal_accuracy: 0.8856 - val_precision: 0.9075
Epoch 4/15
160/160 [==============================] - 13s 84ms/step - loss: 0.2924 - cate
gorical_accuracy: 0.8981 - precision: 0.9171 - val_loss: 0.2891 - val_categori
cal_accuracy: 0.8983 - val_precision: 0.9189
Epoch 5/15
160/160 [==============================] - 14s 86ms/step - loss: 0.2589 - cate
gorical_accuracy: 0.9077 - precision: 0.9257 - val_loss: 0.2782 - val_categori
cal_accuracy: 0.9021 - val_precision: 0.9186
Epoch 6/15
160/160 [==============================] - 22s 136ms/step - loss: 0.2464 - cat
egorical_accuracy: 0.9117 - precision: 0.9280 - val_loss: 0.2889 - val_categor
ical_accuracy: 0.8984 - val_precision: 0.9140
Epoch 7/15
160/160 [==============================] - 14s 85ms/step - loss: 0.2349 - cate
gorical_accuracy: 0.9178 - precision: 0.9325 - val_loss: 0.2601 - val_categori
cal_accuracy: 0.9067 - val_precision: 0.9215
Epoch 8/15
160/160 [==============================] - 13s 84ms/step - loss: 0.2165 - cate
gorical_accuracy: 0.9229 - precision: 0.9364 - val_loss: 0.2645 - val_categori
cal_accuracy: 0.9078 - val_precision: 0.9202
Epoch 9/15
160/160 [==============================] - 14s 84ms/step - loss: 0.2065 - cate
gorical_accuracy: 0.9248 - precision: 0.9361 - val_loss: 0.2615 - val_categori
cal_accuracy: 0.9058 - val_precision: 0.9206
Epoch 10/15
160/160 [==============================] - 14s 86ms/step - loss: 0.1891 - cate
gorical_accuracy: 0.9343 - precision: 0.9448 - val_loss: 0.2559 - val_categori
cal_accuracy: 0.9081 - val_precision: 0.9208
Epoch 11/15
160/160 [==============================] - 14s 85ms/step - loss: 0.1820 - cate
gorical_accuracy: 0.9349 - precision: 0.9449 - val_loss: 0.2497 - val_categori
cal_accuracy: 0.9112 - val_precision: 0.9230
Epoch 12/15
160/160 [==============================] - 14s 85ms/step - loss: 0.1736 - cate
gorical_accuracy: 0.9380 - precision: 0.9482 - val_loss: 0.2432 - val_categori
cal_accuracy: 0.9157 - val_precision: 0.9264
Epoch 13/15
160/160 [==============================] - 13s 84ms/step - loss: 0.1636 - cate
gorical_accuracy: 0.9415 - precision: 0.9503 - val_loss: 0.2534 - val_categori
cal_accuracy: 0.9119 - val_precision: 0.9228
Epoch 14/15
160/160 [==============================] - 13s 83ms/step - loss: 0.1568 - cate
gorical_accuracy: 0.9441 - precision: 0.9514 - val_loss: 0.2464 - val_categori
cal_accuracy: 0.9162 - val_precision: 0.9262
Epoch 15/15
160/160 [==============================] - 26s 163ms/step - loss: 0.1515 - cat
```

```
egorical_accuracy: 0.9472 - precision: 0.9552 - val_loss: 0.2494 - val_categor
ical_accuracy: 0.9141 - val_precision: 0.9242
INFO:tensorflow:Assets written to: cnn/cnn.1/assets
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 32)        832       
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 32)        0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 14, 32)        25632     
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 7, 7, 32)          0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 1568)              0         
_________________________________________________________________
dense_2 (Dense)              (None, 64)                100416    
_________________________________________________________________
dense_3 (Dense)              (None, 10)                650       
=================================================================
Total params: 127,530
Trainable params: 127,530
Non-trainable params: 0
_________________________________________________________________
Epoch 1/15
160/160 [==============================] - 57s 350ms/step - loss: 0.7804 - cat
egorical_accuracy: 0.8171 - precision: 0.8886 - val_loss: 0.3802 - val_categor
ical_accuracy: 0.8616 - val_precision: 0.8848
Epoch 2/15
160/160 [==============================] - 55s 342ms/step - loss: 0.3193 - cat
egorical_accuracy: 0.8838 - precision: 0.9059 - val_loss: 0.3048 - val_categor
ical_accuracy: 0.8913 - val_precision: 0.9128
Epoch 3/15
160/160 [==============================] - 39s 244ms/step - loss: 0.2774 - cat
egorical_accuracy: 0.8992 - precision: 0.9158 - val_loss: 0.2751 - val_categor
ical_accuracy: 0.9016 - val_precision: 0.9172
Epoch 4/15
160/160 [==============================] - 39s 245ms/step - loss: 0.2523 - cat
egorical_accuracy: 0.9079 - precision: 0.9236 - val_loss: 0.2688 - val_categor
ical_accuracy: 0.9054 - val_precision: 0.9196
Epoch 5/15
160/160 [==============================] - 41s 260ms/step - loss: 0.2244 - cat
egorical_accuracy: 0.9171 - precision: 0.9302 - val_loss: 0.2507 - val_categor
ical_accuracy: 0.9094 - val_precision: 0.9224
Epoch 6/15
160/160 [==============================] - 39s 244ms/step - loss: 0.2041 - cat
egorical_accuracy: 0.9253 - precision: 0.9372 - val_loss: 0.2485 - val_categor
ical_accuracy: 0.9126 - val_precision: 0.9251
Epoch 7/15
160/160 [==============================] - 29s 178ms/step - loss: 0.1954 - cat
egorical_accuracy: 0.9258 - precision: 0.9378 - val_loss: 0.2405 - val_categor
ical_accuracy: 0.9134 - val_precision: 0.9225
Epoch 8/15
160/160 [==============================] - 22s 136ms/step - loss: 0.1832 - cat
egorical_accuracy: 0.9324 - precision: 0.9419 - val_loss: 0.2369 - val_categor
ical_accuracy: 0.9128 - val_precision: 0.9240
Epoch 9/15
160/160 [==============================] - 23s 146ms/step - loss: 0.1796 - cat
egorical_accuracy: 0.9348 - precision: 0.9433 - val_loss: 0.2648 - val_categor
ical_accuracy: 0.9045 - val_precision: 0.9133
Epoch 10/15
160/160 [==============================] - 24s 150ms/step - loss: 0.1662 - cat
egorical_accuracy: 0.9417 - precision: 0.9487 - val_loss: 0.2256 - val_categor
ical_accuracy: 0.9191 - val_precision: 0.9278
Epoch 11/15
160/160 [==============================] - 23s 141ms/step - loss: 0.1594 - cat
egorical_accuracy: 0.9420 - precision: 0.9496 - val_loss: 0.2301 - val_categor
ical_accuracy: 0.9183 - val_precision: 0.9270
```

```
Epoch 12/15
160/160 [==============================] - 22s 141ms/step - loss: 0.1459 - cat
egorical_accuracy: 0.9476 - precision: 0.9534 - val_loss: 0.2281 - val_categor
ical_accuracy: 0.9207 - val_precision: 0.9290
Epoch 13/15
160/160 [==============================] - 23s 141ms/step - loss: 0.1374 - cat
egorical_accuracy: 0.9496 - precision: 0.9553 - val_loss: 0.2305 - val_categor
ical_accuracy: 0.9199 - val_precision: 0.9274
Epoch 14/15
160/160 [==============================] - 26s 166ms/step - loss: 0.1327 - cat
egorical_accuracy: 0.9530 - precision: 0.9581 - val_loss: 0.2263 - val_categor
ical_accuracy: 0.9224 - val_precision: 0.9285
Epoch 15/15
160/160 [==============================] - 22s 140ms/step - loss: 0.1244 - cat
egorical_accuracy: 0.9566 - precision: 0.9616 - val_loss: 0.2237 - val_categor
ical_accuracy: 0.9210 - val_precision: 0.9277
INFO:tensorflow:Assets written to: cnn/cnn.2/assets
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_3 (Conv2D)            (None, 28, 28, 32)        832       
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 14, 14, 32)        0         
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 14, 14, 32)        25632     
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 7, 7, 32)          0         
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 7, 7, 32)          25632     
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 3, 3, 32)          0         
_________________________________________________________________
flatten_2 (Flatten)          (None, 288)               0         
_________________________________________________________________
dense_4 (Dense)              (None, 64)                18496     
_________________________________________________________________
dense_5 (Dense)              (None, 10)                650       
=================================================================
Total params: 71,242
Trainable params: 71,242
Non-trainable params: 0
_________________________________________________________________
Epoch 1/15
160/160 [==============================] - 42s 262ms/step - loss: 0.9682 - cat
egorical_accuracy: 0.7841 - precision: 0.8791 - val_loss: 0.3903 - val_categor
ical_accuracy: 0.8576 - val_precision: 0.8937
Epoch 2/15
160/160 [==============================] - 26s 164ms/step - loss: 0.3520 - cat
egorical_accuracy: 0.8723 - precision: 0.8997 - val_loss: 0.3287 - val_categor
ical_accuracy: 0.8794 - val_precision: 0.8990
Epoch 3/15
160/160 [==============================] - 34s 216ms/step - loss: 0.2959 - cat
egorical_accuracy: 0.8912 - precision: 0.9111 - val_loss: 0.3096 - val_categor
ical_accuracy: 0.8850 - val_precision: 0.9026
Epoch 4/15
160/160 [==============================] - 26s 163ms/step - loss: 0.2641 - cat
egorical_accuracy: 0.9050 - precision: 0.9227 - val_loss: 0.2855 - val_categor
ical_accuracy: 0.8973 - val_precision: 0.9166
Epoch 5/15
160/160 [==============================] - 26s 163ms/step - loss: 0.2445 - cat
egorical_accuracy: 0.9109 - precision: 0.9269 - val_loss: 0.2884 - val_categor
ical_accuracy: 0.8953 - val_precision: 0.9119
Epoch 6/15
160/160 [==============================] - 26s 163ms/step - loss: 0.2291 - cat
egorical_accuracy: 0.9149 - precision: 0.9281 - val_loss: 0.2564 - val_categor
ical_accuracy: 0.9055 - val_precision: 0.9227
Epoch 7/15
160/160 [==============================] - 26s 165ms/step - loss: 0.2097 - cat
```

```
egorical_accuracy: 0.9233 - precision: 0.9349 - val_loss: 0.2478 - val_categor
ical_accuracy: 0.9068 - val_precision: 0.9211
Epoch 8/15
160/160 [==============================] - 26s 165ms/step - loss: 0.1980 - cat
egorical_accuracy: 0.9287 - precision: 0.9397 - val_loss: 0.2454 - val_categor
ical_accuracy: 0.9112 - val_precision: 0.9237
Epoch 9/15
160/160 [==============================] - 27s 168ms/step - loss: 0.1879 - cat
egorical_accuracy: 0.9315 - precision: 0.9410 - val_loss: 0.2401 - val_categor
ical_accuracy: 0.9125 - val_precision: 0.9255
Epoch 10/15
160/160 [==============================] - 26s 165ms/step - loss: 0.1789 - cat
egorical_accuracy: 0.9342 - precision: 0.9431 - val_loss: 0.2411 - val_categor
ical_accuracy: 0.9112 - val_precision: 0.9240
Epoch 11/15
160/160 [==============================] - 28s 173ms/step - loss: 0.1693 - cat
egorical_accuracy: 0.9384 - precision: 0.9466 - val_loss: 0.2477 - val_categor
ical_accuracy: 0.9103 - val_precision: 0.9218
Epoch 12/15
160/160 [==============================] - 36s 223ms/step - loss: 0.1603 - cat
egorical_accuracy: 0.9426 - precision: 0.9508 - val_loss: 0.2332 - val_categor
ical_accuracy: 0.9156 - val_precision: 0.9265
Epoch 13/15
160/160 [==============================] - 27s 169ms/step - loss: 0.1501 - cat
egorical_accuracy: 0.9466 - precision: 0.9533 - val_loss: 0.2323 - val_categor
ical_accuracy: 0.9162 - val_precision: 0.9268
Epoch 14/15
160/160 [==============================] - 26s 162ms/step - loss: 0.1418 - cat
egorical_accuracy: 0.9497 - precision: 0.9561 - val_loss: 0.2383 - val_categor
ical_accuracy: 0.9166 - val_precision: 0.9246
Epoch 15/15
160/160 [==============================] - 29s 183ms/step - loss: 0.1389 - cat
egorical_accuracy: 0.9494 - precision: 0.9554 - val_loss: 0.2436 - val_categor
ical_accuracy: 0.9134 - val_precision: 0.9207
INFO:tensorflow:Assets written to: cnn/cnn.3/assets
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_6 (Conv2D)            (None, 28, 28, 32)        832       
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 14, 14, 32)        0         
_________________________________________________________________
flatten_3 (Flatten)          (None, 6272)              0         
_________________________________________________________________
dense_6 (Dense)              (None, 64)                401472    
_________________________________________________________________
dense_7 (Dense)              (None, 64)                4160      
_________________________________________________________________
dense_8 (Dense)              (None, 10)                650       
=================================================================
Total params: 407,114
Trainable params: 407,114
Non-trainable params: 0
_________________________________________________________________
Epoch 1/15
160/160 [==============================] - 21s 127ms/step - loss: 0.7719 - cat
egorical_accuracy: 0.8226 - precision: 0.8886 - val_loss: 0.3381 - val_categor
ical_accuracy: 0.8798 - val_precision: 0.9078
Epoch 2/15
160/160 [==============================] - 20s 122ms/step - loss: 0.3019 - cat
egorical_accuracy: 0.8906 - precision: 0.9146 - val_loss: 0.2927 - val_categor
ical_accuracy: 0.8962 - val_precision: 0.9146
Epoch 3/15
160/160 [==============================] - 26s 162ms/step - loss: 0.2596 - cat
egorical_accuracy: 0.9046 - precision: 0.9215 - val_loss: 0.2696 - val_categor
ical_accuracy: 0.9032 - val_precision: 0.9209
Epoch 4/15
160/160 [==============================] - 20s 127ms/step - loss: 0.2274 - cat
```

```
egorical_accuracy: 0.9158 - precision: 0.9295 - val_loss: 0.2755 - val_categor
ical_accuracy: 0.9020 - val_precision: 0.9180
Epoch 5/15
160/160 [==============================] - 12s 75ms/step - loss: 0.2042 - cate
gorical_accuracy: 0.9266 - precision: 0.9386 - val_loss: 0.2564 - val_categori
cal_accuracy: 0.9094 - val_precision: 0.9222
Epoch 6/15
160/160 [==============================] - 9s 59ms/step - loss: 0.1895 - categ
orical_accuracy: 0.9313 - precision: 0.9421 - val_loss: 0.2496 - val_categoric
al_accuracy: 0.9103 - val_precision: 0.9223
Epoch 7/15
160/160 [==============================] - 14s 90ms/step - loss: 0.1758 - cate
gorical_accuracy: 0.9369 - precision: 0.9462 - val_loss: 0.2440 - val_categori
cal_accuracy: 0.9144 - val_precision: 0.9257
Epoch 8/15
160/160 [==============================] - 15s 91ms/step - loss: 0.1572 - cate
gorical_accuracy: 0.9450 - precision: 0.9535 - val_loss: 0.2486 - val_categori
cal_accuracy: 0.9127 - val_precision: 0.9228
Epoch 9/15
160/160 [==============================] - 14s 90ms/step - loss: 0.1464 - cate
gorical_accuracy: 0.9476 - precision: 0.9563 - val_loss: 0.2549 - val_categori
cal_accuracy: 0.9117 - val_precision: 0.9200
Epoch 10/15
160/160 [==============================] - 14s 91ms/step - loss: 0.1355 - cate
gorical_accuracy: 0.9522 - precision: 0.9577 - val_loss: 0.2484 - val_categori
cal_accuracy: 0.9138 - val_precision: 0.9213
Epoch 11/15
160/160 [==============================] - 15s 92ms/step - loss: 0.1224 - cate
gorical_accuracy: 0.9583 - precision: 0.9639 - val_loss: 0.2614 - val_categori
cal_accuracy: 0.9128 - val_precision: 0.9207
INFO:tensorflow:Assets written to: cnn/cnn.4/assets
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_7 (Conv2D)            (None, 28, 28, 32)        832       
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 14, 14, 32)        0         
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 14, 14, 32)        25632     
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 7, 7, 32)          0         
_________________________________________________________________
flatten_4 (Flatten)          (None, 1568)              0         
_________________________________________________________________
dense_9 (Dense)              (None, 64)                100416    
_________________________________________________________________
dense_10 (Dense)             (None, 64)                4160      
_________________________________________________________________
dense_11 (Dense)             (None, 10)                650       
=================================================================
Total params: 131,690
Trainable params: 131,690
Non-trainable params: 0
_________________________________________________________________
Epoch 1/15
160/160 [==============================] - 59s 355ms/step - loss: 0.9402 - cat
egorical_accuracy: 0.7831 - precision: 0.8724 - val_loss: 0.3933 - val_categor
ical_accuracy: 0.8563 - val_precision: 0.8887
Epoch 2/15
160/160 [==============================] - 44s 277ms/step - loss: 0.3532 - cat
egorical_accuracy: 0.8703 - precision: 0.9003 - val_loss: 0.3241 - val_categor
ical_accuracy: 0.8826 - val_precision: 0.9052
Epoch 3/15
160/160 [==============================] - 45s 280ms/step - loss: 0.2945 - cat
egorical_accuracy: 0.8918 - precision: 0.9107 - val_loss: 0.2882 - val_categor
ical_accuracy: 0.8950 - val_precision: 0.9147
Epoch 4/15
160/160 [==============================] - 46s 286ms/step - loss: 0.2620 - cat
```

```
egorical_accuracy: 0.9034 - precision: 0.9207 - val_loss: 0.2738 - val_categor
ical_accuracy: 0.8991 - val_precision: 0.9169
Epoch 5/15
160/160 [==============================] - 43s 266ms/step - loss: 0.2401 - cat
egorical_accuracy: 0.9094 - precision: 0.9260 - val_loss: 0.2532 - val_categor
ical_accuracy: 0.9062 - val_precision: 0.9230
Epoch 6/15
160/160 [==============================] - 44s 273ms/step - loss: 0.2254 - cat
egorical_accuracy: 0.9171 - precision: 0.9304 - val_loss: 0.2439 - val_categor
ical_accuracy: 0.9112 - val_precision: 0.9254
Epoch 7/15
160/160 [==============================] - 50s 315ms/step - loss: 0.2159 - cat
egorical_accuracy: 0.9196 - precision: 0.9324 - val_loss: 0.2496 - val_categor
ical_accuracy: 0.9079 - val_precision: 0.9208
Epoch 8/15
160/160 [==============================] - 43s 272ms/step - loss: 0.2031 - cat
egorical_accuracy: 0.9265 - precision: 0.9377 - val_loss: 0.2467 - val_categor
ical_accuracy: 0.9098 - val_precision: 0.9225
Epoch 9/15
160/160 [==============================] - 45s 283ms/step - loss: 0.1895 - cat
egorical_accuracy: 0.9304 - precision: 0.9409 - val_loss: 0.2401 - val_categor
ical_accuracy: 0.9122 - val_precision: 0.9263
Epoch 10/15
160/160 [==============================] - 45s 282ms/step - loss: 0.1838 - cat
egorical_accuracy: 0.9313 - precision: 0.9420 - val_loss: 0.2361 - val_categor
ical_accuracy: 0.9134 - val_precision: 0.9252
Epoch 11/15
160/160 [==============================] - 45s 280ms/step - loss: 0.1746 - cat
egorical_accuracy: 0.9360 - precision: 0.9449 - val_loss: 0.2352 - val_categor
ical_accuracy: 0.9128 - val_precision: 0.9247
Epoch 12/15
160/160 [==============================] - 45s 279ms/step - loss: 0.1638 - cat
egorical_accuracy: 0.9408 - precision: 0.9496 - val_loss: 0.2424 - val_categor
ical_accuracy: 0.9119 - val_precision: 0.9231
Epoch 13/15
160/160 [==============================] - 43s 269ms/step - loss: 0.1615 - cat
egorical_accuracy: 0.9425 - precision: 0.9497 - val_loss: 0.2332 - val_categor
ical_accuracy: 0.9172 - val_precision: 0.9256
Epoch 14/15
160/160 [==============================] - 41s 259ms/step - loss: 0.1447 - cat
egorical_accuracy: 0.9468 - precision: 0.9549 - val_loss: 0.2307 - val_categor
ical_accuracy: 0.9174 - val_precision: 0.9272
Epoch 15/15
160/160 [==============================] - 41s 254ms/step - loss: 0.1427 - cat
egorical_accuracy: 0.9481 - precision: 0.9554 - val_loss: 0.2232 - val_categor
ical_accuracy: 0.9195 - val_precision: 0.9300
INFO:tensorflow:Assets written to: cnn/cnn.5/assets
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_9 (Conv2D)            (None, 28, 28, 32)        832
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 14, 14, 32)        25632
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 7, 7, 32)          0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 7, 7, 32)          25632
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 3, 3, 32)          0
_________________________________________________________________
flatten_5 (Flatten)          (None, 288)               0
_________________________________________________________________
dense_12 (Dense)             (None, 64)                18496
_________________________________________________________________
dense_13 (Dense)             (None, 64)                4160
_________________________________________________________________
```

```
dense_14 (Dense)             (None, 10)                650
=================================================================
Total params: 75,402
Trainable params: 75,402
Non-trainable params: 0
_________________________________________________________________
Epoch 1/15
160/160 [==============================] - 92s 565ms/step - loss: 1.0592 - cat
egorical_accuracy: 0.7670 - precision: 0.8816 - val_loss: 0.4163 - val_categor
ical_accuracy: 0.8469 - val_precision: 0.8840
Epoch 2/15
160/160 [==============================] - 46s 288ms/step - loss: 0.3967 - cat
egorical_accuracy: 0.8568 - precision: 0.8900 - val_loss: 0.3505 - val_categor
ical_accuracy: 0.8708 - val_precision: 0.8990
Epoch 3/15
160/160 [==============================] - 49s 304ms/step - loss: 0.3203 - cat
egorical_accuracy: 0.8789 - precision: 0.9033 - val_loss: 0.3266 - val_categor
ical_accuracy: 0.8823 - val_precision: 0.9029
Epoch 4/15
160/160 [==============================] - 49s 308ms/step - loss: 0.2849 - cat
egorical_accuracy: 0.8960 - precision: 0.9152 - val_loss: 0.3072 - val_categor
ical_accuracy: 0.8852 - val_precision: 0.9064
Epoch 5/15
160/160 [==============================] - 50s 309ms/step - loss: 0.2685 - cat
egorical_accuracy: 0.9015 - precision: 0.9187 - val_loss: 0.2910 - val_categor
ical_accuracy: 0.8909 - val_precision: 0.9103
Epoch 6/15
160/160 [==============================] - 50s 312ms/step - loss: 0.2596 - cat
egorical_accuracy: 0.9033 - precision: 0.9187 - val_loss: 0.2693 - val_categor
ical_accuracy: 0.9022 - val_precision: 0.9174
Epoch 7/15
160/160 [==============================] - 49s 306ms/step - loss: 0.2353 - cat
egorical_accuracy: 0.9143 - precision: 0.9281 - val_loss: 0.2644 - val_categor
ical_accuracy: 0.9053 - val_precision: 0.9203
Epoch 8/15
160/160 [==============================] - 51s 320ms/step - loss: 0.2206 - cat
egorical_accuracy: 0.9201 - precision: 0.9330 - val_loss: 0.2605 - val_categor
ical_accuracy: 0.9081 - val_precision: 0.9217
Epoch 9/15
160/160 [==============================] - 50s 312ms/step - loss: 0.2073 - cat
egorical_accuracy: 0.9243 - precision: 0.9364 - val_loss: 0.2599 - val_categor
ical_accuracy: 0.9078 - val_precision: 0.9204
Epoch 10/15
160/160 [==============================] - 49s 306ms/step - loss: 0.2009 - cat
egorical_accuracy: 0.9252 - precision: 0.9375 - val_loss: 0.2506 - val_categor
ical_accuracy: 0.9103 - val_precision: 0.9222
Epoch 11/15
160/160 [==============================] - 50s 310ms/step - loss: 0.1902 - cat
egorical_accuracy: 0.9299 - precision: 0.9411 - val_loss: 0.2544 - val_categor
ical_accuracy: 0.9104 - val_precision: 0.9207
Epoch 12/15
160/160 [==============================] - 51s 317ms/step - loss: 0.1776 - cat
egorical_accuracy: 0.9324 - precision: 0.9428 - val_loss: 0.2595 - val_categor
ical_accuracy: 0.9066 - val_precision: 0.9197
Epoch 13/15
160/160 [==============================] - 51s 320ms/step - loss: 0.1727 - cat
egorical_accuracy: 0.9363 - precision: 0.9447 - val_loss: 0.2517 - val_categor
ical_accuracy: 0.9117 - val_precision: 0.9224
Epoch 14/15
160/160 [==============================] - 54s 335ms/step - loss: 0.1624 - cat
egorical_accuracy: 0.9405 - precision: 0.9492 - val_loss: 0.2714 - val_categor
ical_accuracy: 0.9047 - val_precision: 0.9140
Epoch 15/15
160/160 [==============================] - 50s 315ms/step - loss: 0.1594 - cat
egorical_accuracy: 0.9414 - precision: 0.9492 - val_loss: 0.2455 - val_categor
ical_accuracy: 0.9122 - val_precision: 0.9203
INFO:tensorflow:Assets written to: cnn/cnn.6/assets
```

CNN training performance

Now the training performances can be visualised

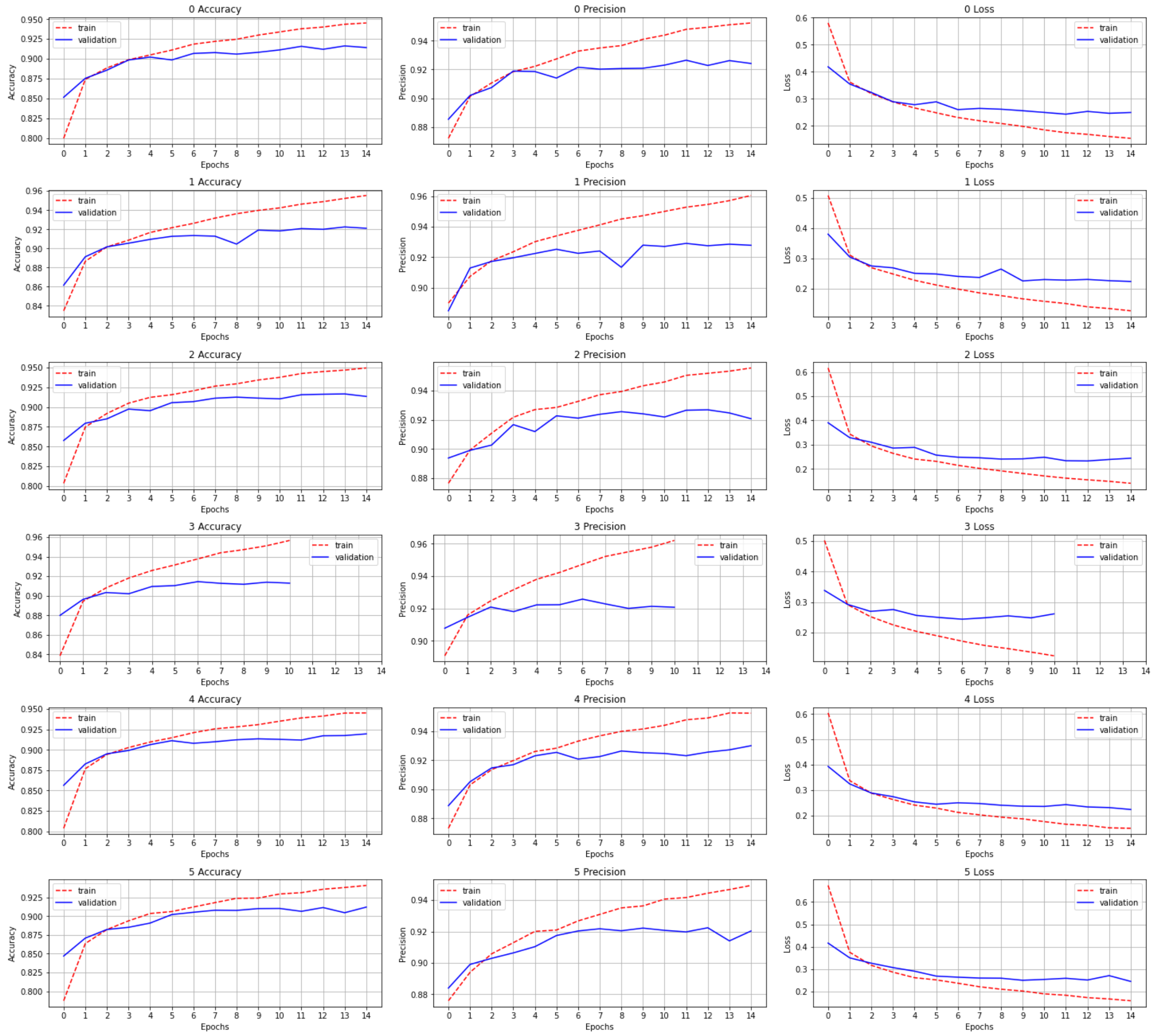
In [12]:

```
def visualise(count, historys):
    fig, axs = plt.subplots(count, 3, figsize=(20, 18), constrained_layout=Tr

    metrics = [('categorical_accuracy', 'val_categorical_accuracy'), ('precis
    titles = ['Accuracy', 'Precision', 'Loss']

    for idx, ax in enumerate(axs):
        for i in range(3):
            ax[i].plot(historys[idx].history[metrics[i][0]], label='train', l
            ax[i].plot(historys[idx].history[metrics[i][1]], label='validatio
            ax[i].set_title((f'{idx} {titles[i]}'))
            ax[i].set_ylabel(titles[i])
            ax[i].set_xlabel('Epochs')
            ax[i].set_xticks([x for x in np.arange(15)])
            ax[i].legend()
            ax[i].grid()
```

From these graphs, we a trying to find a model that has a validation accuracy or precision moving above the train accuracy or precision or a validation loss moving below the train loss. The reason for this is because it implies that the model is either able to or learning how to generalise well instead of learning the training data.

In [13]:

```
visualise(cnn_count, cnn_history)
```



CNN refactoring

The CNN architecture I chose to improve was the CNN.N1 model, this was because this model outperformed all of the other models over a longer period of time making it the most consistent throughout the training and validation stages of the training phase.

From looking at the graphs above we can see that each of models testing 'Categorical Accuracy' and 'Precision' are scoring lower than the training 'Categorical Accuracy' and 'Precision' and the 'Loss' is also higher when the model is testing. From this we can tell that the model is overfitting. Overfitting is where the model familiarises itself with the training set and is subsequently unable to generalise well. This is caused by noise in the training data that the network picks up during training and learns it as an underlying concept of the data. To counter this people employ a regularisation technique.

Regularisation can be defined in this context as a set of different techniques that lower the complexity of a neural network model during training. There are 3 main regularisation techniques to employ, l1 regularisation, l2 regularisation and dropout. Where l1 regularisation forces the weight parameters to become 0, and l2 regularisation forces the weight parameters towards 0 (but never 0).

I have chosen to use a Dropout layer to counter the overfitting. In a Dropout layer, you are able to select a percentage of the number of neurons in the model you want to lose during training. For example, I will be applying a dropout rate of 0.3, this means that during training

30% of the neurons in the network will randomly have their weights set to 0, meaning they will be lost from the network. The loss of neurons to dropout is done at each forward propagation and weight update step.

Below is a function that places a Dropout layer in a certain position based upon the number provided as place. This was done to find the optimal location to place the Dropout layer. This could be refactored to have as many place holders as you want active Dropout layers.

In [14]:

```
def dropcnn(place):
    model = Sequential()
    # input layer
    model.add(Input(shape=INPUT_SHAPE))
    # 3 convolution & pooling layers
    model.add(Conv2D(CONV_UNITS, kernel_size=KERNEL_SIZE, activation=ACTIV[0]
    model.add(MaxPool2D(pool_size=POOL_SIZE))
    if place == 1:
        model.add(Dropout(DROPOUT))
    model.add(Conv2D(CONV_UNITS, kernel_size=KERNEL_SIZE, activation=ACTIV[0]
    model.add(MaxPool2D(pool_size=POOL_SIZE))
    if place == 2:
        model.add(Dropout(DROPOUT))
    # flatten to 1 dimension
    model.add(Flatten())
    if place == 3:
        model.add(Dropout(DROPOUT))
    # fully connected layer
    model.add(Dense(FULLY_UNITS, activation=ACTIV[0]))
    if place == 4:
        model.add(Dropout(DROPOUT))
    # output layer
    model.add(Dense(NUM_CLASSES, activation=ACTIV[1]))
    return model
```

In [15]:

```
dropcnn_count, dropcnn_history = training(dropcnn, 'dropcnn')
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_12 (Conv2D)           (None, 28, 28, 32)        832       
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 14, 14, 32)        0         
_________________________________________________________________
dropout (Dropout)            (None, 14, 14, 32)        0         
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 14, 14, 32)        25632     
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 7, 7, 32)          0         
_________________________________________________________________
flatten_6 (Flatten)          (None, 1568)              0         
_________________________________________________________________
dense_15 (Dense)             (None, 64)                100416    
_________________________________________________________________
dense_16 (Dense)             (None, 10)                650       
=================================================================
Total params: 127,530
Trainable params: 127,530
Non-trainable params: 0
_________________________________________________________________
Epoch 1/15
160/160 [==============================] - 81s 500ms/step - loss: 0.8461 - cat
egorical_accuracy: 0.8038 - precision: 0.8771 - val_loss: 0.3834 - val_categor
```

```
ical_accuracy: 0.8625 - val_precision: 0.8914
Epoch 2/15
160/160 [==============================] - 59s 367ms/step - loss: 0.3644 - cat
egorical_accuracy: 0.8691 - precision: 0.8947 - val_loss: 0.3271 - val_categor
ical_accuracy: 0.8838 - val_precision: 0.9057
Epoch 3/15
160/160 [==============================] - 46s 285ms/step - loss: 0.3165 - cat
egorical_accuracy: 0.8858 - precision: 0.9058 - val_loss: 0.3190 - val_categor
ical_accuracy: 0.8876 - val_precision: 0.9119
Epoch 4/15
160/160 [==============================] - 46s 287ms/step - loss: 0.2992 - cat
egorical_accuracy: 0.8914 - precision: 0.9110 - val_loss: 0.2833 - val_categor
ical_accuracy: 0.9007 - val_precision: 0.9181
Epoch 5/15
160/160 [==============================] - 45s 283ms/step - loss: 0.2812 - cat
egorical_accuracy: 0.8958 - precision: 0.9141 - val_loss: 0.2684 - val_categor
ical_accuracy: 0.9050 - val_precision: 0.9236
Epoch 6/15
160/160 [==============================] - 47s 295ms/step - loss: 0.2695 - cat
egorical_accuracy: 0.9014 - precision: 0.9180 - val_loss: 0.2654 - val_categor
ical_accuracy: 0.9046 - val_precision: 0.9213
Epoch 7/15
160/160 [==============================] - 45s 283ms/step - loss: 0.2490 - cat
egorical_accuracy: 0.9084 - precision: 0.9236 - val_loss: 0.2553 - val_categor
ical_accuracy: 0.9095 - val_precision: 0.9256
Epoch 8/15
160/160 [==============================] - 52s 325ms/step - loss: 0.2483 - cat
egorical_accuracy: 0.9077 - precision: 0.9220 - val_loss: 0.2718 - val_categor
ical_accuracy: 0.9000 - val_precision: 0.9150
Epoch 9/15
160/160 [==============================] - 49s 308ms/step - loss: 0.2423 - cat
egorical_accuracy: 0.9107 - precision: 0.9258 - val_loss: 0.2442 - val_categor
ical_accuracy: 0.9111 - val_precision: 0.9242
Epoch 10/15
160/160 [==============================] - 50s 311ms/step - loss: 0.2320 - cat
egorical_accuracy: 0.9141 - precision: 0.9270 - val_loss: 0.2396 - val_categor
ical_accuracy: 0.9121 - val_precision: 0.9268
Epoch 11/15
160/160 [==============================] - 50s 312ms/step - loss: 0.2197 - cat
egorical_accuracy: 0.9200 - precision: 0.9310 - val_loss: 0.2345 - val_categor
ical_accuracy: 0.9156 - val_precision: 0.9294
Epoch 12/15
160/160 [==============================] - 49s 306ms/step - loss: 0.2118 - cat
egorical_accuracy: 0.9209 - precision: 0.9327 - val_loss: 0.2403 - val_categor
ical_accuracy: 0.9131 - val_precision: 0.9268
Epoch 13/15
160/160 [==============================] - 47s 297ms/step - loss: 0.2114 - cat
egorical_accuracy: 0.9211 - precision: 0.9322 - val_loss: 0.2348 - val_categor
ical_accuracy: 0.9148 - val_precision: 0.9268
Epoch 14/15
160/160 [==============================] - 41s 258ms/step - loss: 0.2126 - cat
egorical_accuracy: 0.9216 - precision: 0.9339 - val_loss: 0.2271 - val_categor
ical_accuracy: 0.9194 - val_precision: 0.9315
Epoch 15/15
160/160 [==============================] - 39s 243ms/step - loss: 0.1988 - cat
egorical_accuracy: 0.9261 - precision: 0.9363 - val_loss: 0.2331 - val_categor
ical_accuracy: 0.9164 - val_precision: 0.9290
INFO:tensorflow:Assets written to: dropcnn/dropcnn.1/assets
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_14 (Conv2D)           (None, 28, 28, 32)        832       
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 14, 14, 32)        0         
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 14, 14, 32)        25632     
_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 7, 7, 32)          0         
```

```
_________________________________________________________________
dropout_1 (Dropout)          (None, 7, 7, 32)          0
_________________________________________________________________
flatten_7 (Flatten)          (None, 1568)              0
_________________________________________________________________
dense_17 (Dense)             (None, 64)                100416
_________________________________________________________________
dense_18 (Dense)             (None, 10)                650
=================================================================
Total params: 127,530
Trainable params: 127,530
Non-trainable params: 0
_________________________________________________________________
Epoch 1/15
160/160 [==============================] - 55s 335ms/step - loss: 0.8742 - cat
egorical_accuracy: 0.7953 - precision: 0.8753 - val_loss: 0.3580 - val_categor
ical_accuracy: 0.8704 - val_precision: 0.9001
Epoch 2/15
160/160 [==============================] - 66s 413ms/step - loss: 0.3732 - cat
egorical_accuracy: 0.8655 - precision: 0.8939 - val_loss: 0.3093 - val_categor
ical_accuracy: 0.8901 - val_precision: 0.9193
Epoch 3/15
160/160 [==============================] - 37s 228ms/step - loss: 0.3312 - cat
egorical_accuracy: 0.8778 - precision: 0.9018 - val_loss: 0.2825 - val_categor
ical_accuracy: 0.8972 - val_precision: 0.9189
Epoch 4/15
160/160 [==============================] - 36s 222ms/step - loss: 0.2983 - cat
egorical_accuracy: 0.8909 - precision: 0.9101 - val_loss: 0.2698 - val_categor
ical_accuracy: 0.9013 - val_precision: 0.9171
Epoch 5/15
160/160 [==============================] - 38s 239ms/step - loss: 0.2778 - cat
egorical_accuracy: 0.8962 - precision: 0.9147 - val_loss: 0.2594 - val_categor
ical_accuracy: 0.9036 - val_precision: 0.9227
Epoch 6/15
160/160 [==============================] - 37s 229ms/step - loss: 0.2706 - cat
egorical_accuracy: 0.8988 - precision: 0.9165 - val_loss: 0.2497 - val_categor
ical_accuracy: 0.9087 - val_precision: 0.9257
Epoch 7/15
160/160 [==============================] - 36s 227ms/step - loss: 0.2561 - cat
egorical_accuracy: 0.9052 - precision: 0.9206 - val_loss: 0.2603 - val_categor
ical_accuracy: 0.9043 - val_precision: 0.9220
Epoch 8/15
160/160 [==============================] - 37s 229ms/step - loss: 0.2491 - cat
egorical_accuracy: 0.9067 - precision: 0.9218 - val_loss: 0.2355 - val_categor
ical_accuracy: 0.9134 - val_precision: 0.9285
Epoch 9/15
160/160 [==============================] - 36s 228ms/step - loss: 0.2431 - cat
egorical_accuracy: 0.9106 - precision: 0.9245 - val_loss: 0.2329 - val_categor
ical_accuracy: 0.9125 - val_precision: 0.9273
Epoch 10/15
160/160 [==============================] - 37s 230ms/step - loss: 0.2295 - cat
egorical_accuracy: 0.9162 - precision: 0.9286 - val_loss: 0.2295 - val_categor
ical_accuracy: 0.9150 - val_precision: 0.9288
Epoch 11/15
160/160 [==============================] - 36s 228ms/step - loss: 0.2251 - cat
egorical_accuracy: 0.9154 - precision: 0.9292 - val_loss: 0.2249 - val_categor
ical_accuracy: 0.9163 - val_precision: 0.9285
Epoch 12/15
160/160 [==============================] - 36s 227ms/step - loss: 0.2150 - cat
egorical_accuracy: 0.9191 - precision: 0.9315 - val_loss: 0.2234 - val_categor
ical_accuracy: 0.9168 - val_precision: 0.9293
Epoch 13/15
160/160 [==============================] - 36s 228ms/step - loss: 0.2062 - cat
egorical_accuracy: 0.9221 - precision: 0.9344 - val_loss: 0.2220 - val_categor
ical_accuracy: 0.9178 - val_precision: 0.9298
Epoch 14/15
160/160 [==============================] - 36s 224ms/step - loss: 0.2097 - cat
egorical_accuracy: 0.9223 - precision: 0.9337 - val_loss: 0.2202 - val_categor
ical_accuracy: 0.9183 - val_precision: 0.9300
```

```
Epoch 15/15
160/160 [==============================] - 36s 223ms/step - loss: 0.2019 - cat
egorical_accuracy: 0.9258 - precision: 0.9378 - val_loss: 0.2220 - val_categor
ical_accuracy: 0.9174 - val_precision: 0.9280
INFO:tensorflow:Assets written to: dropcnn/dropcnn.2/assets
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_16 (Conv2D)           (None, 28, 28, 32)        832
_________________________________________________________________
max_pooling2d_16 (MaxPooling (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 14, 14, 32)        25632
_________________________________________________________________
max_pooling2d_17 (MaxPooling (None, 7, 7, 32)          0
_________________________________________________________________
flatten_8 (Flatten)          (None, 1568)              0
_________________________________________________________________
dropout_2 (Dropout)          (None, 1568)              0
_________________________________________________________________
dense_19 (Dense)             (None, 64)                100416
_________________________________________________________________
dense_20 (Dense)             (None, 10)                650
=================================================================
Total params: 127,530
Trainable params: 127,530
Non-trainable params: 0
_________________________________________________________________
Epoch 1/15
160/160 [==============================] - 90s 554ms/step - loss: 0.8403 - cat
egorical_accuracy: 0.8052 - precision: 0.8788 - val_loss: 0.3775 - val_categor
ical_accuracy: 0.8608 - val_precision: 0.8924
Epoch 2/15
160/160 [==============================] - 53s 333ms/step - loss: 0.3792 - cat
egorical_accuracy: 0.8620 - precision: 0.8918 - val_loss: 0.3129 - val_categor
ical_accuracy: 0.8900 - val_precision: 0.9173
Epoch 3/15
160/160 [==============================] - 43s 267ms/step - loss: 0.3324 - cat
egorical_accuracy: 0.8782 - precision: 0.9012 - val_loss: 0.2878 - val_categor
ical_accuracy: 0.8956 - val_precision: 0.9144
Epoch 4/15
160/160 [==============================] - 36s 227ms/step - loss: 0.3041 - cat
egorical_accuracy: 0.8868 - precision: 0.9073 - val_loss: 0.2830 - val_categor
ical_accuracy: 0.8972 - val_precision: 0.9189
Epoch 5/15
160/160 [==============================] - 37s 228ms/step - loss: 0.2851 - cat
egorical_accuracy: 0.8955 - precision: 0.9136 - val_loss: 0.2588 - val_categor
ical_accuracy: 0.9057 - val_precision: 0.9201
Epoch 6/15
160/160 [==============================] - 37s 233ms/step - loss: 0.2622 - cat
egorical_accuracy: 0.9036 - precision: 0.9195 - val_loss: 0.2562 - val_categor
ical_accuracy: 0.9053 - val_precision: 0.9210
Epoch 7/15
160/160 [==============================] - 37s 233ms/step - loss: 0.2511 - cat
egorical_accuracy: 0.9058 - precision: 0.9212 - val_loss: 0.2447 - val_categor
ical_accuracy: 0.9129 - val_precision: 0.9283
Epoch 8/15
160/160 [==============================] - 37s 231ms/step - loss: 0.2525 - cat
egorical_accuracy: 0.9063 - precision: 0.9215 - val_loss: 0.2436 - val_categor
ical_accuracy: 0.9120 - val_precision: 0.9266
Epoch 9/15
160/160 [==============================] - 38s 241ms/step - loss: 0.2389 - cat
egorical_accuracy: 0.9111 - precision: 0.9253 - val_loss: 0.2338 - val_categor
ical_accuracy: 0.9133 - val_precision: 0.9262
Epoch 10/15
160/160 [==============================] - 41s 256ms/step - loss: 0.2332 - cat
egorical_accuracy: 0.9138 - precision: 0.9282 - val_loss: 0.2330 - val_categor
ical_accuracy: 0.9164 - val_precision: 0.9281
```

```
Epoch 11/15
160/160 [==============================] - 41s 259ms/step - loss: 0.2238 - cat
egorical_accuracy: 0.9154 - precision: 0.9282 - val_loss: 0.2306 - val_categor
ical_accuracy: 0.9169 - val_precision: 0.9278
Epoch 12/15
160/160 [==============================] - 41s 257ms/step - loss: 0.2174 - cat
egorical_accuracy: 0.9168 - precision: 0.9299 - val_loss: 0.2334 - val_categor
ical_accuracy: 0.9147 - val_precision: 0.9250
Epoch 13/15
160/160 [==============================] - 41s 256ms/step - loss: 0.2191 - cat
egorical_accuracy: 0.9192 - precision: 0.9310 - val_loss: 0.2235 - val_categor
ical_accuracy: 0.9184 - val_precision: 0.9296
Epoch 14/15
160/160 [==============================] - 46s 288ms/step - loss: 0.2037 - cat
egorical_accuracy: 0.9244 - precision: 0.9341 - val_loss: 0.2188 - val_categor
ical_accuracy: 0.9205 - val_precision: 0.9327
Epoch 15/15
160/160 [==============================] - 44s 278ms/step - loss: 0.2084 - cat
egorical_accuracy: 0.9221 - precision: 0.9326 - val_loss: 0.2221 - val_categor
ical_accuracy: 0.9207 - val_precision: 0.9295
INFO:tensorflow:Assets written to: dropcnn/dropcnn.3/assets
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_18 (Conv2D)           (None, 28, 28, 32)        832       
_________________________________________________________________
max_pooling2d_18 (MaxPooling (None, 14, 14, 32)        0         
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 14, 14, 32)        25632     
_________________________________________________________________
max_pooling2d_19 (MaxPooling (None, 7, 7, 32)          0         
_________________________________________________________________
flatten_9 (Flatten)          (None, 1568)              0         
_________________________________________________________________
dense_21 (Dense)             (None, 64)                100416    
_________________________________________________________________
dropout_3 (Dropout)          (None, 64)                0         
_________________________________________________________________
dense_22 (Dense)             (None, 10)                650       
=================================================================
Total params: 127,530
Trainable params: 127,530
Non-trainable params: 0
_________________________________________________________________
Epoch 1/15
160/160 [==============================] - 60s 362ms/step - loss: 0.9543 - cat
egorical_accuracy: 0.7819 - precision: 0.8820 - val_loss: 0.3870 - val_categor
ical_accuracy: 0.8565 - val_precision: 0.9013
Epoch 2/15
160/160 [==============================] - 46s 286ms/step - loss: 0.4572 - cat
egorical_accuracy: 0.8364 - precision: 0.8912 - val_loss: 0.3338 - val_categor
ical_accuracy: 0.8773 - val_precision: 0.9123
Epoch 3/15
160/160 [==============================] - 45s 282ms/step - loss: 0.3924 - cat
egorical_accuracy: 0.8602 - precision: 0.9012 - val_loss: 0.3108 - val_categor
ical_accuracy: 0.8848 - val_precision: 0.9105
Epoch 4/15
160/160 [==============================] - 48s 301ms/step - loss: 0.3584 - cat
egorical_accuracy: 0.8676 - precision: 0.9058 - val_loss: 0.2923 - val_categor
ical_accuracy: 0.8934 - val_precision: 0.9122
Epoch 5/15
160/160 [==============================] - 44s 272ms/step - loss: 0.3326 - cat
egorical_accuracy: 0.8815 - precision: 0.9130 - val_loss: 0.2802 - val_categor
ical_accuracy: 0.8975 - val_precision: 0.9182
Epoch 6/15
160/160 [==============================] - 39s 245ms/step - loss: 0.3268 - cat
egorical_accuracy: 0.8833 - precision: 0.9150 - val_loss: 0.2652 - val_categor
ical_accuracy: 0.9035 - val_precision: 0.9227
```

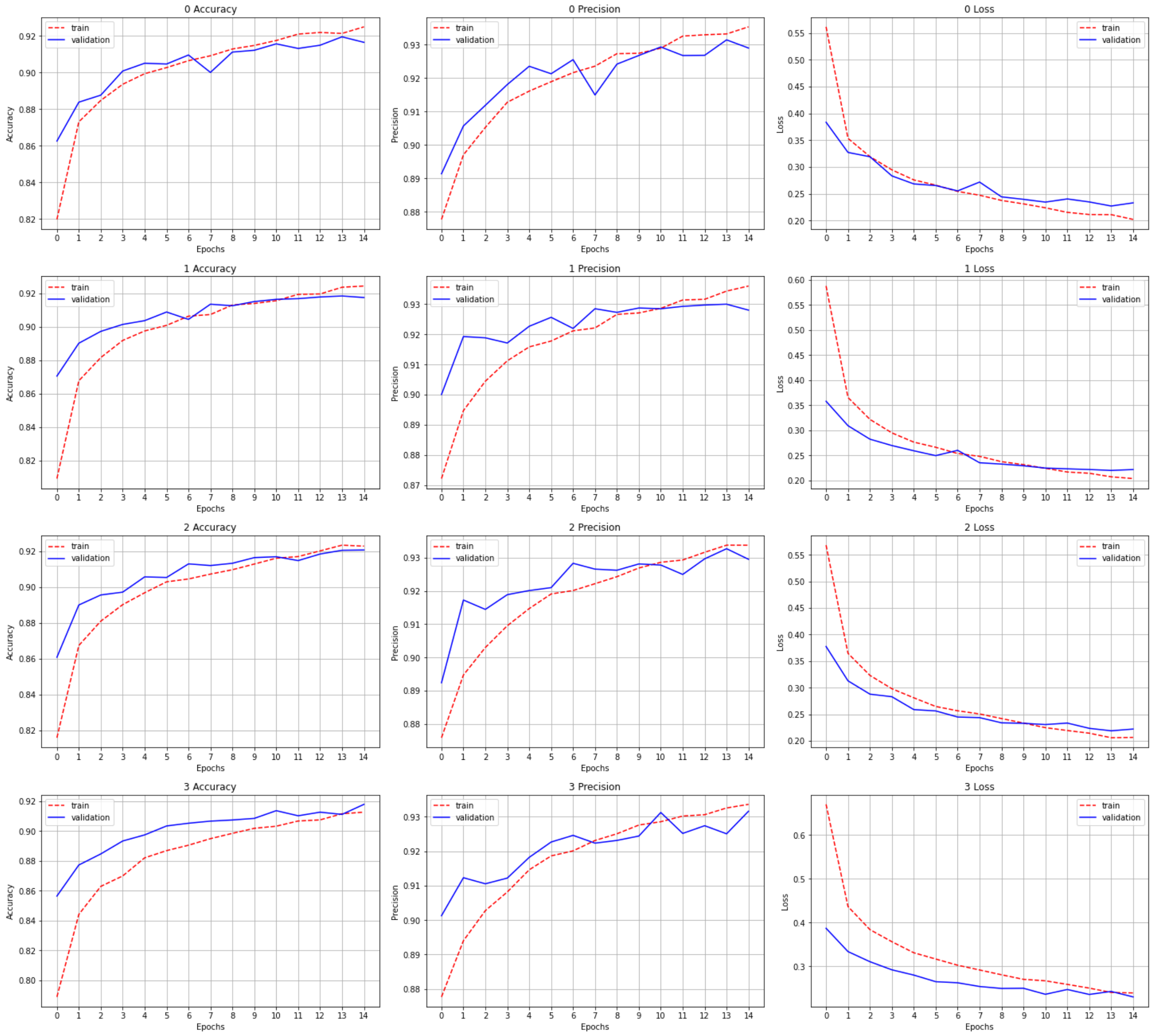
```
Epoch 7/15
160/160 [==============================] - 39s 241ms/step - loss: 0.3028 - cat
egorical_accuracy: 0.8913 - precision: 0.9211 - val_loss: 0.2627 - val_categor
ical_accuracy: 0.9053 - val_precision: 0.9246
Epoch 8/15
160/160 [==============================] - 40s 252ms/step - loss: 0.2898 - cat
egorical_accuracy: 0.8963 - precision: 0.9247 - val_loss: 0.2542 - val_categor
ical_accuracy: 0.9068 - val_precision: 0.9224
Epoch 9/15
160/160 [==============================] - 43s 267ms/step - loss: 0.2780 - cat
egorical_accuracy: 0.8991 - precision: 0.9246 - val_loss: 0.2497 - val_categor
ical_accuracy: 0.9075 - val_precision: 0.9231
Epoch 10/15
160/160 [==============================] - 41s 255ms/step - loss: 0.2699 - cat
egorical_accuracy: 0.9012 - precision: 0.9259 - val_loss: 0.2502 - val_categor
ical_accuracy: 0.9086 - val_precision: 0.9244
Epoch 11/15
160/160 [==============================] - 34s 214ms/step - loss: 0.2662 - cat
egorical_accuracy: 0.9036 - precision: 0.9289 - val_loss: 0.2364 - val_categor
ical_accuracy: 0.9137 - val_precision: 0.9313
Epoch 12/15
160/160 [==============================] - 21s 134ms/step - loss: 0.2571 - cat
egorical_accuracy: 0.9071 - precision: 0.9309 - val_loss: 0.2474 - val_categor
ical_accuracy: 0.9103 - val_precision: 0.9252
Epoch 13/15
160/160 [==============================] - 20s 128ms/step - loss: 0.2515 - cat
egorical_accuracy: 0.9077 - precision: 0.9314 - val_loss: 0.2361 - val_categor
ical_accuracy: 0.9128 - val_precision: 0.9274
Epoch 14/15
160/160 [==============================] - 20s 128ms/step - loss: 0.2407 - cat
egorical_accuracy: 0.9126 - precision: 0.9332 - val_loss: 0.2429 - val_categor
ical_accuracy: 0.9112 - val_precision: 0.9251
Epoch 15/15
160/160 [==============================] - 20s 128ms/step - loss: 0.2385 - cat
egorical_accuracy: 0.9138 - precision: 0.9346 - val_loss: 0.2304 - val_categor
ical_accuracy: 0.9180 - val_precision: 0.9316
INFO:tensorflow:Assets written to: dropcnn/dropcnn.4/assets
```

CNN refactored performance

Now that the models have less nodes in their network, in theory they should be able generalise better. However these examples below are still not the optimal model. The architecture that recieved the best scores will be put forward to have the parameters of the model tuned, for the best performance. After which, the model will be tested and have its scored compared to that of the FCNN model.

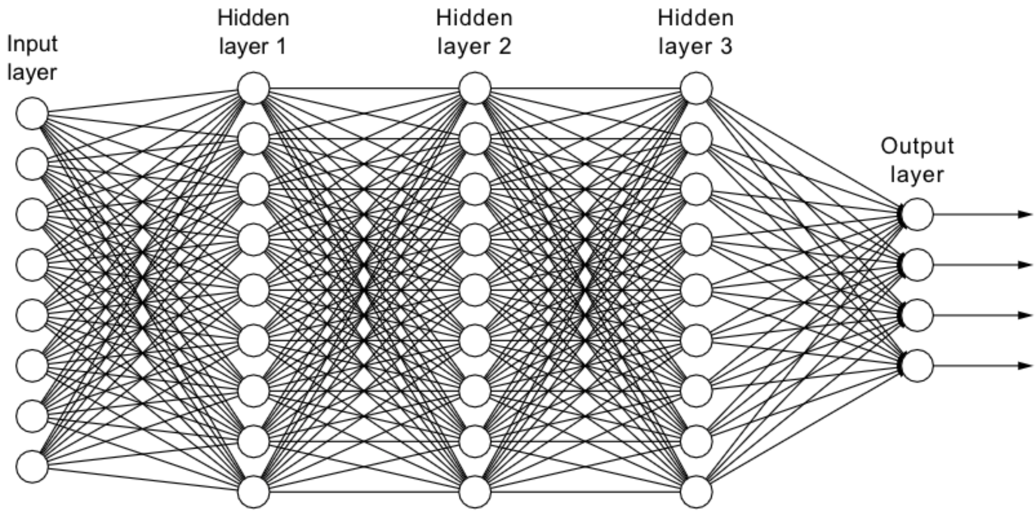
In [16]:

```
visualise(dropcnn_count, dropcnn_history)
```



What is a Fully Connected Neural Network

'Fully connected neural networks (FCNNs) are a type of artificial neural network where the architecture is such that all the nodes or neurones, in one layer are connected to the neurones in the next layer.' (Moore, 2019)



(Dürr, Stick and Murina, 2020)

FCNN architecture

The below function creates different FCNN models, where there is another fully connected layer added after the flatten layer each time a model has been trained. Until the model has 4 fully connected layers after the flatten layer.

In [17]:

```
def fcnn_arch(n):
    model = Sequential()
    model.add(Input(INPUT_SHAPE))
    model.add(Dense(FULLY_UNITS, activation=ACTIV[0]))
    model.add(Flatten())
    for _ in np.arange(n):
        model.add(Dense(FULLY_UNITS, activation=ACTIV[0]))
    model.add(Dense(NUM_CLASSES, activation=ACTIV[1]))
    return model
```

FCNN training

Here the above function is used to create 4 different FCNN models and each of these models are trained on the training data. The purpose of this is to find the optimal number of fully connected layers to add after the flatten layer.

In [18]:

```
fcnn_count, fcnn_history = training(fcnn_arch, 'fcnn')
```

```
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_23 (Dense)             (None, 28, 28, 64)        128       
_________________________________________________________________
flatten_10 (Flatten)         (None, 50176)             0         
_________________________________________________________________
dense_24 (Dense)             (None, 64)                3211328   
_________________________________________________________________
dense_25 (Dense)             (None, 10)                650       
=================================================================
Total params: 3,212,106
Trainable params: 3,212,106
Non-trainable params: 0
_________________________________________________________________
Epoch 1/15
160/160 [==============================] - 17s 106ms/step - loss: 1.4100 - cat
egorical_accuracy: 0.7856 - precision: 0.8653 - val_loss: 0.4737 - val_categor
ical_accuracy: 0.8342 - val_precision: 0.8781
Epoch 2/15
160/160 [==============================] - 14s 90ms/step - loss: 0.4460 - cate
gorical_accuracy: 0.8471 - precision: 0.8849 - val_loss: 0.4254 - val_categori
cal_accuracy: 0.8525 - val_precision: 0.8863
Epoch 3/15
160/160 [==============================] - 13s 84ms/step - loss: 0.4227 - cate
gorical_accuracy: 0.8533 - precision: 0.8856 - val_loss: 0.4040 - val_categori
cal_accuracy: 0.8571 - val_precision: 0.8888
Epoch 4/15
160/160 [==============================] - 12s 76ms/step - loss: 0.3930 - cate
gorical_accuracy: 0.8623 - precision: 0.8937 - val_loss: 0.3943 - val_categori
cal_accuracy: 0.8626 - val_precision: 0.8920
Epoch 5/15
160/160 [==============================] - 12s 77ms/step - loss: 0.3692 - cate
gorical_accuracy: 0.8704 - precision: 0.8973 - val_loss: 0.3932 - val_categori
cal_accuracy: 0.8585 - val_precision: 0.8864
Epoch 6/15
160/160 [==============================] - 12s 77ms/step - loss: 0.3608 - cate
gorical_accuracy: 0.8720 - precision: 0.8965 - val_loss: 0.3854 - val_categori
```

```
cal_accuracy: 0.8603 - val_precision: 0.8869
Epoch 7/15
160/160 [==============================] - 13s 80ms/step - loss: 0.3563 - cate
gorical_accuracy: 0.8732 - precision: 0.8974 - val_loss: 0.3825 - val_categori
cal_accuracy: 0.8636 - val_precision: 0.8887
Epoch 8/15
160/160 [==============================] - 15s 97ms/step - loss: 0.3490 - cate
gorical_accuracy: 0.8750 - precision: 0.8978 - val_loss: 0.3683 - val_categori
cal_accuracy: 0.8696 - val_precision: 0.8967
Epoch 9/15
160/160 [==============================] - 13s 79ms/step - loss: 0.3386 - cate
gorical_accuracy: 0.8773 - precision: 0.8998 - val_loss: 0.3633 - val_categori
cal_accuracy: 0.8709 - val_precision: 0.8929
Epoch 10/15
160/160 [==============================] - 12s 74ms/step - loss: 0.3215 - cate
gorical_accuracy: 0.8832 - precision: 0.9057 - val_loss: 0.3813 - val_categori
cal_accuracy: 0.8637 - val_precision: 0.8852
Epoch 11/15
160/160 [==============================] - 12s 75ms/step - loss: 0.3159 - cate
gorical_accuracy: 0.8864 - precision: 0.9084 - val_loss: 0.3538 - val_categori
cal_accuracy: 0.8735 - val_precision: 0.8978
Epoch 12/15
160/160 [==============================] - 12s 75ms/step - loss: 0.3072 - cate
gorical_accuracy: 0.8883 - precision: 0.9091 - val_loss: 0.3549 - val_categori
cal_accuracy: 0.8719 - val_precision: 0.8944
Epoch 13/15
160/160 [==============================] - 11s 70ms/step - loss: 0.3048 - cate
gorical_accuracy: 0.8905 - precision: 0.9112 - val_loss: 0.3559 - val_categori
cal_accuracy: 0.8762 - val_precision: 0.8949
Epoch 14/15
160/160 [==============================] - 12s 74ms/step - loss: 0.2963 - cate
gorical_accuracy: 0.8927 - precision: 0.9123 - val_loss: 0.3539 - val_categori
cal_accuracy: 0.8737 - val_precision: 0.8930
Epoch 15/15
160/160 [==============================] - 12s 76ms/step - loss: 0.2894 - cate
gorical_accuracy: 0.8948 - precision: 0.9130 - val_loss: 0.3579 - val_categori
cal_accuracy: 0.8762 - val_precision: 0.8934
INFO:tensorflow:Assets written to: fcnn/fcnn.1/assets
Model: "sequential_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_26 (Dense)             (None, 28, 28, 64)        128       
_________________________________________________________________
flatten_11 (Flatten)         (None, 50176)             0         
_________________________________________________________________
dense_27 (Dense)             (None, 64)                3211328   
_________________________________________________________________
dense_28 (Dense)             (None, 64)                4160      
_________________________________________________________________
dense_29 (Dense)             (None, 10)                650       
=================================================================
Total params: 3,216,266
Trainable params: 3,216,266
Non-trainable params: 0
_________________________________________________________________
Epoch 1/15
160/160 [==============================] - 26s 161ms/step - loss: 0.9732 - cat
egorical_accuracy: 0.7727 - precision: 0.8515 - val_loss: 0.4722 - val_categor
ical_accuracy: 0.8301 - val_precision: 0.8687
Epoch 2/15
160/160 [==============================] - 12s 74ms/step - loss: 0.4177 - cate
gorical_accuracy: 0.8510 - precision: 0.8845 - val_loss: 0.4014 - val_categori
cal_accuracy: 0.8571 - val_precision: 0.8903
Epoch 3/15
160/160 [==============================] - 11s 71ms/step - loss: 0.3732 - cate
gorical_accuracy: 0.8632 - precision: 0.8933 - val_loss: 0.3861 - val_categori
cal_accuracy: 0.8613 - val_precision: 0.8927
Epoch 4/15
```

```
160/160 [==============================] - 12s 73ms/step - loss: 0.3449 - cate
gorical_accuracy: 0.8762 - precision: 0.9032 - val_loss: 0.3666 - val_categori
cal_accuracy: 0.8677 - val_precision: 0.8938
Epoch 5/15
160/160 [==============================] - 11s 72ms/step - loss: 0.3288 - cate
gorical_accuracy: 0.8805 - precision: 0.9068 - val_loss: 0.3614 - val_categori
cal_accuracy: 0.8685 - val_precision: 0.8974
Epoch 6/15
160/160 [==============================] - 12s 73ms/step - loss: 0.3238 - cate
gorical_accuracy: 0.8803 - precision: 0.9049 - val_loss: 0.3517 - val_categori
cal_accuracy: 0.8726 - val_precision: 0.8976
Epoch 7/15
160/160 [==============================] - 12s 74ms/step - loss: 0.3019 - cate
gorical_accuracy: 0.8886 - precision: 0.9119 - val_loss: 0.3491 - val_categori
cal_accuracy: 0.8718 - val_precision: 0.8964
Epoch 8/15
160/160 [==============================] - 12s 73ms/step - loss: 0.3032 - cate
gorical_accuracy: 0.8862 - precision: 0.9095 - val_loss: 0.3523 - val_categori
cal_accuracy: 0.8745 - val_precision: 0.8963
Epoch 9/15
160/160 [==============================] - 11s 72ms/step - loss: 0.2963 - cate
gorical_accuracy: 0.8921 - precision: 0.9133 - val_loss: 0.3493 - val_categori
cal_accuracy: 0.8746 - val_precision: 0.8969
Epoch 10/15
160/160 [==============================] - 12s 72ms/step - loss: 0.2876 - cate
gorical_accuracy: 0.8937 - precision: 0.9144 - val_loss: 0.3453 - val_categori
cal_accuracy: 0.8734 - val_precision: 0.8960
Epoch 11/15
160/160 [==============================] - 11s 72ms/step - loss: 0.2834 - cate
gorical_accuracy: 0.8947 - precision: 0.9159 - val_loss: 0.3390 - val_categori
cal_accuracy: 0.8792 - val_precision: 0.9009
Epoch 12/15
160/160 [==============================] - 11s 72ms/step - loss: 0.2733 - cate
gorical_accuracy: 0.8980 - precision: 0.9179 - val_loss: 0.3604 - val_categori
cal_accuracy: 0.8758 - val_precision: 0.8945
Epoch 13/15
160/160 [==============================] - 11s 71ms/step - loss: 0.2636 - cate
gorical_accuracy: 0.9035 - precision: 0.9220 - val_loss: 0.3490 - val_categori
cal_accuracy: 0.8788 - val_precision: 0.8970
Epoch 14/15
160/160 [==============================] - 11s 72ms/step - loss: 0.2623 - cate
gorical_accuracy: 0.9027 - precision: 0.9204 - val_loss: 0.3451 - val_categori
cal_accuracy: 0.8787 - val_precision: 0.8965
Epoch 15/15
160/160 [==============================] - 11s 71ms/step - loss: 0.2601 - cate
gorical_accuracy: 0.9022 - precision: 0.9196 - val_loss: 0.3391 - val_categori
cal_accuracy: 0.8817 - val_precision: 0.9006
INFO:tensorflow:Assets written to: fcnn/fcnn.2/assets
Model: "sequential_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_30 (Dense)             (None, 28, 28, 64)        128       
_________________________________________________________________
flatten_12 (Flatten)         (None, 50176)             0         
_________________________________________________________________
dense_31 (Dense)             (None, 64)                3211328   
_________________________________________________________________
dense_32 (Dense)             (None, 64)                4160      
_________________________________________________________________
dense_33 (Dense)             (None, 64)                4160      
_________________________________________________________________
dense_34 (Dense)             (None, 10)                650       
=================================================================
Total params: 3,220,426
Trainable params: 3,220,426
Non-trainable params: 0
_________________________________________________________________
Epoch 1/15
```

```
160/160 [==============================] - 26s 161ms/step - loss: 1.0894 - cat
egorical_accuracy: 0.7654 - precision: 0.8525 - val_loss: 0.4620 - val_categor
ical_accuracy: 0.8371 - val_precision: 0.8775
Epoch 2/15
160/160 [==============================] - 28s 175ms/step - loss: 0.4579 - cat
egorical_accuracy: 0.8377 - precision: 0.8745 - val_loss: 0.4074 - val_categor
ical_accuracy: 0.8541 - val_precision: 0.8833
Epoch 3/15
160/160 [==============================] - 18s 114ms/step - loss: 0.3938 - cat
egorical_accuracy: 0.8599 - precision: 0.8896 - val_loss: 0.3938 - val_categor
ical_accuracy: 0.8587 - val_precision: 0.8917
Epoch 4/15
160/160 [==============================] - 12s 73ms/step - loss: 0.3701 - cate
gorical_accuracy: 0.8654 - precision: 0.8940 - val_loss: 0.3751 - val_categori
cal_accuracy: 0.8653 - val_precision: 0.8920
Epoch 5/15
160/160 [==============================] - 12s 73ms/step - loss: 0.3477 - cate
gorical_accuracy: 0.8717 - precision: 0.8982 - val_loss: 0.3677 - val_categori
cal_accuracy: 0.8695 - val_precision: 0.8947
Epoch 6/15
160/160 [==============================] - 12s 74ms/step - loss: 0.3334 - cate
gorical_accuracy: 0.8758 - precision: 0.9005 - val_loss: 0.3751 - val_categori
cal_accuracy: 0.8625 - val_precision: 0.8913
Epoch 7/15
160/160 [==============================] - 12s 73ms/step - loss: 0.3313 - cate
gorical_accuracy: 0.8783 - precision: 0.9030 - val_loss: 0.3642 - val_categori
cal_accuracy: 0.8706 - val_precision: 0.8951
Epoch 8/15
160/160 [==============================] - 12s 74ms/step - loss: 0.3212 - cate
gorical_accuracy: 0.8835 - precision: 0.9068 - val_loss: 0.3580 - val_categori
cal_accuracy: 0.8687 - val_precision: 0.8963
Epoch 9/15
160/160 [==============================] - 12s 77ms/step - loss: 0.3083 - cate
gorical_accuracy: 0.8861 - precision: 0.9106 - val_loss: 0.3655 - val_categori
cal_accuracy: 0.8697 - val_precision: 0.8915
Epoch 10/15
160/160 [==============================] - 12s 76ms/step - loss: 0.2956 - cate
gorical_accuracy: 0.8907 - precision: 0.9120 - val_loss: 0.3394 - val_categori
cal_accuracy: 0.8782 - val_precision: 0.8997
Epoch 11/15
160/160 [==============================] - 12s 74ms/step - loss: 0.2864 - cate
gorical_accuracy: 0.8928 - precision: 0.9113 - val_loss: 0.3461 - val_categori
cal_accuracy: 0.8761 - val_precision: 0.8971
Epoch 12/15
160/160 [==============================] - 12s 74ms/step - loss: 0.2859 - cate
gorical_accuracy: 0.8929 - precision: 0.9136 - val_loss: 0.3484 - val_categori
cal_accuracy: 0.8774 - val_precision: 0.8981
Epoch 13/15
160/160 [==============================] - 12s 74ms/step - loss: 0.2802 - cate
gorical_accuracy: 0.8961 - precision: 0.9183 - val_loss: 0.3494 - val_categori
cal_accuracy: 0.8745 - val_precision: 0.8950
Epoch 14/15
160/160 [==============================] - 12s 75ms/step - loss: 0.2706 - cate
gorical_accuracy: 0.8984 - precision: 0.9187 - val_loss: 0.3474 - val_categori
cal_accuracy: 0.8742 - val_precision: 0.8987
INFO:tensorflow:Assets written to: fcnn/fcnn.3/assets
Model: "sequential_13"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_35 (Dense)             (None, 28, 28, 64)        128       
_________________________________________________________________
flatten_13 (Flatten)         (None, 50176)             0         
_________________________________________________________________
dense_36 (Dense)             (None, 64)                3211328   
_________________________________________________________________
dense_37 (Dense)             (None, 64)                4160      
_________________________________________________________________
dense_38 (Dense)             (None, 64)                4160      
```

```
_________________________________________________________________
dense_39 (Dense)             (None, 64)                4160      
_________________________________________________________________
dense_40 (Dense)             (None, 10)                650       
=================================================================
Total params: 3,224,586
Trainable params: 3,224,586
Non-trainable params: 0

_________________________________________________________________
Epoch 1/15
160/160 [==============================] - 24s 150ms/step - loss: 0.9887 - cat
egorical_accuracy: 0.7596 - precision: 0.8538 - val_loss: 0.4549 - val_categor
ical_accuracy: 0.8343 - val_precision: 0.8790
Epoch 2/15
160/160 [==============================] - 24s 150ms/step - loss: 0.4376 - cat
egorical_accuracy: 0.8413 - precision: 0.8806 - val_loss: 0.4303 - val_categor
ical_accuracy: 0.8469 - val_precision: 0.8807
Epoch 3/15
160/160 [==============================] - 12s 76ms/step - loss: 0.3863 - cate
gorical_accuracy: 0.8592 - precision: 0.8924 - val_loss: 0.3839 - val_categori
cal_accuracy: 0.8583 - val_precision: 0.8907
Epoch 4/15
160/160 [==============================] - 12s 75ms/step - loss: 0.3611 - cate
gorical_accuracy: 0.8673 - precision: 0.8975 - val_loss: 0.3741 - val_categori
cal_accuracy: 0.8653 - val_precision: 0.8925
Epoch 5/15
160/160 [==============================] - 12s 78ms/step - loss: 0.3369 - cate
gorical_accuracy: 0.8768 - precision: 0.9044 - val_loss: 0.3643 - val_categori
cal_accuracy: 0.8677 - val_precision: 0.8993
Epoch 6/15
160/160 [==============================] - 12s 76ms/step - loss: 0.3257 - cate
gorical_accuracy: 0.8799 - precision: 0.9073 - val_loss: 0.3625 - val_categori
cal_accuracy: 0.8733 - val_precision: 0.8998
Epoch 7/15
160/160 [==============================] - 12s 75ms/step - loss: 0.3078 - cate
gorical_accuracy: 0.8868 - precision: 0.9106 - val_loss: 0.3612 - val_categori
cal_accuracy: 0.8704 - val_precision: 0.8956
Epoch 8/15
160/160 [==============================] - 12s 74ms/step - loss: 0.3052 - cate
gorical_accuracy: 0.8861 - precision: 0.9115 - val_loss: 0.3692 - val_categori
cal_accuracy: 0.8679 - val_precision: 0.8936
Epoch 9/15
160/160 [==============================] - 12s 74ms/step - loss: 0.2937 - cate
gorical_accuracy: 0.8895 - precision: 0.9128 - val_loss: 0.3473 - val_categori
cal_accuracy: 0.8761 - val_precision: 0.9006
Epoch 10/15
160/160 [==============================] - 12s 75ms/step - loss: 0.2862 - cate
gorical_accuracy: 0.8943 - precision: 0.9149 - val_loss: 0.3423 - val_categori
cal_accuracy: 0.8783 - val_precision: 0.8986
Epoch 11/15
160/160 [==============================] - 12s 75ms/step - loss: 0.2770 - cate
gorical_accuracy: 0.8970 - precision: 0.9187 - val_loss: 0.3514 - val_categori
cal_accuracy: 0.8728 - val_precision: 0.8939
Epoch 12/15
160/160 [==============================] - 12s 74ms/step - loss: 0.2680 - cate
gorical_accuracy: 0.8983 - precision: 0.9195 - val_loss: 0.3462 - val_categori
cal_accuracy: 0.8771 - val_precision: 0.8943
Epoch 13/15
160/160 [==============================] - 12s 73ms/step - loss: 0.2584 - cate
gorical_accuracy: 0.9023 - precision: 0.9206 - val_loss: 0.3380 - val_categori
cal_accuracy: 0.8791 - val_precision: 0.9027
Epoch 14/15
160/160 [==============================] - 12s 74ms/step - loss: 0.2574 - cate
gorical_accuracy: 0.9035 - precision: 0.9220 - val_loss: 0.3399 - val_categori
cal_accuracy: 0.8797 - val_precision: 0.8973
Epoch 15/15
160/160 [==============================] - 12s 74ms/step - loss: 0.2552 - cate
gorical_accuracy: 0.9044 - precision: 0.9220 - val_loss: 0.3802 - val_categori
```

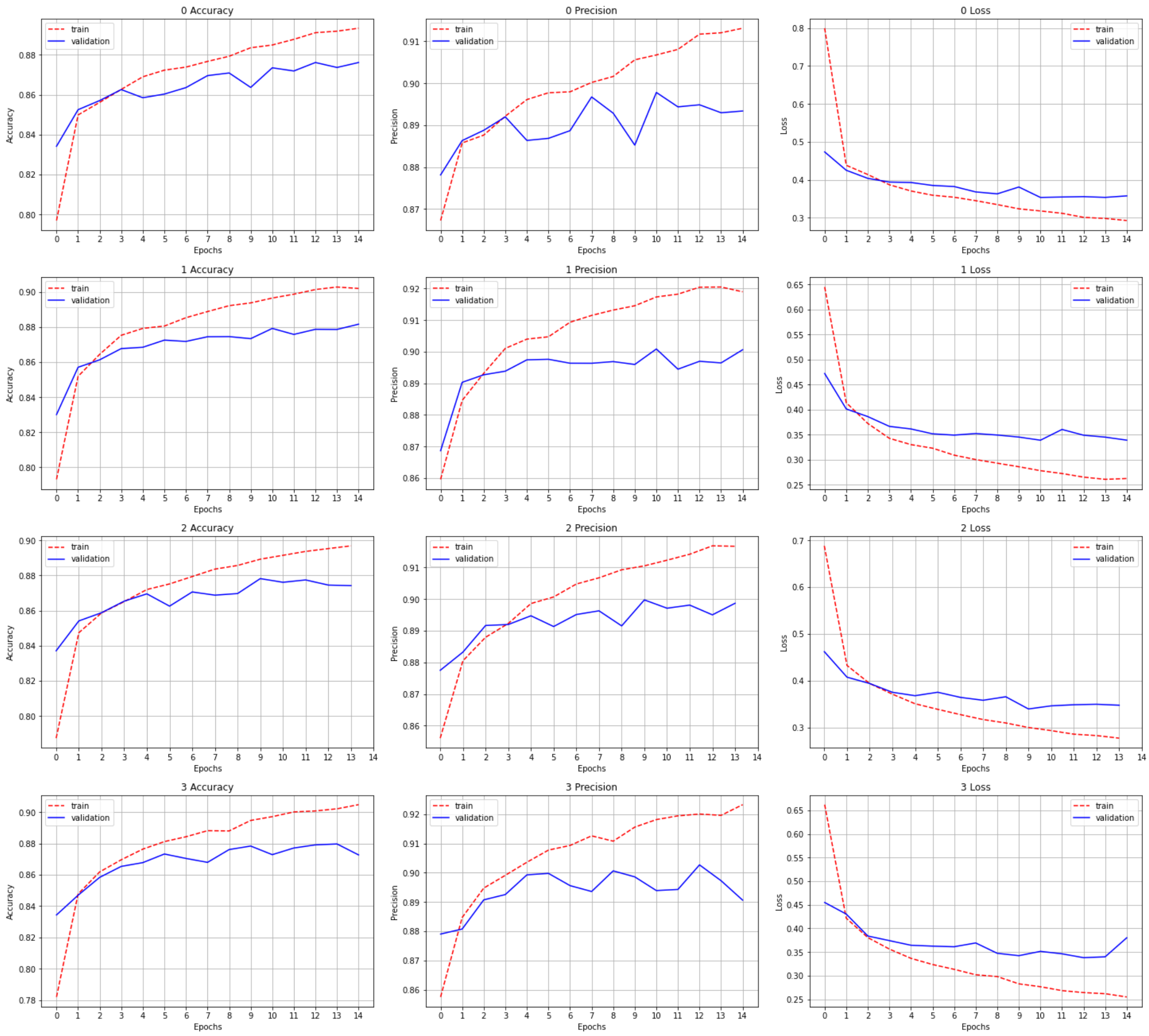
```
cal_accuracy: 0.8727 - val_precision: 0.8906
INFO:tensorflow:Assets written to: fcnn/fcnn.4/assets
```

FCNN training performance

The below graphs show that without a regularisation step the model overfits massively on the training data. With it rarely scoring higher for accuracy and precision or lower for the loss on the validation phase.

In [19]:

```
visualise(fcnn_count, fcnn_history)
```



FCNN refactoring

As seen with the CNN model, the FCNN model is overfitting on the training data. To try and counter this, I am going to employ the same dropout technique as decribed in CNN refactoring.

In [20]:

```
def dropfcnn(place):
    model = Sequential()
    model.add(Input(INPUT_SHAPE))
    model.add(Dense(FULLY_UNITS, activation=ACTIV[0]))
    model.add(Flatten())
    if place == 1:
        model.add(Dropout(DROPOUT))
    model.add(Dense(FULLY_UNITS, activation=ACTIV[0]))
    if place == 2:
```

```
        model.add(Dropout(DROPOUT))
    model.add(Dense(FULLY_UNITS, activation=ACTIV[0]))
    if place == 3:
        model.add(Dropout(DROPOUT))
    model.add(Dense(FULLY_UNITS, activation=ACTIV[0]))
    if place == 4:
        model.add(Dropout(DROPOUT))
    model.add(Dense(NUM_CLASSES, activation=ACTIV[1]))
    return model
```

In [21]:

```
dropfcnn_count, dropfcnn_history = training(dropfcnn, 'dropfcnn')
```

```
Model: "sequential_14"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_41 (Dense)             (None, 28, 28, 64)        128       
_________________________________________________________________
flatten_14 (Flatten)         (None, 50176)             0         
_________________________________________________________________
dropout_4 (Dropout)          (None, 50176)             0         
_________________________________________________________________
dense_42 (Dense)             (None, 64)                3211328   
_________________________________________________________________
dense_43 (Dense)             (None, 64)                4160      
_________________________________________________________________
dense_44 (Dense)             (None, 64)                4160      
_________________________________________________________________
dense_45 (Dense)             (None, 10)                650       
=================================================================
Total params: 3,220,426
Trainable params: 3,220,426
Non-trainable params: 0
_________________________________________________________________
Epoch 1/15
160/160 [==============================] - 45s 281ms/step - loss: 1.0391 - cat
egorical_accuracy: 0.7614 - precision: 0.8502 - val_loss: 0.4472 - val_categor
ical_accuracy: 0.8413 - val_precision: 0.8814
Epoch 2/15
160/160 [==============================] - 20s 124ms/step - loss: 0.4411 - cat
egorical_accuracy: 0.8433 - precision: 0.8803 - val_loss: 0.4048 - val_categor
ical_accuracy: 0.8568 - val_precision: 0.8919
Epoch 3/15
160/160 [==============================] - 20s 123ms/step - loss: 0.4018 - cat
egorical_accuracy: 0.8563 - precision: 0.8875 - val_loss: 0.3791 - val_categor
ical_accuracy: 0.8648 - val_precision: 0.8961
Epoch 4/15
160/160 [==============================] - 20s 124ms/step - loss: 0.3728 - cat
egorical_accuracy: 0.8645 - precision: 0.8922 - val_loss: 0.3728 - val_categor
ical_accuracy: 0.8673 - val_precision: 0.8938
Epoch 5/15
160/160 [==============================] - 20s 123ms/step - loss: 0.3549 - cat
egorical_accuracy: 0.8705 - precision: 0.8989 - val_loss: 0.3676 - val_categor
ical_accuracy: 0.8679 - val_precision: 0.8960
Epoch 6/15
160/160 [==============================] - 20s 125ms/step - loss: 0.3489 - cat
egorical_accuracy: 0.8721 - precision: 0.8988 - val_loss: 0.3621 - val_categor
ical_accuracy: 0.8677 - val_precision: 0.8975
Epoch 7/15
160/160 [==============================] - 20s 126ms/step - loss: 0.3368 - cat
egorical_accuracy: 0.8765 - precision: 0.9021 - val_loss: 0.3570 - val_categor
ical_accuracy: 0.8739 - val_precision: 0.9025
Epoch 8/15
160/160 [==============================] - 20s 124ms/step - loss: 0.3207 - cat
egorical_accuracy: 0.8804 - precision: 0.9053 - val_loss: 0.3449 - val_categor
ical_accuracy: 0.8758 - val_precision: 0.9033
Epoch 9/15
```

```
160/160 [==============================] - 20s 125ms/step - loss: 0.3170 - cat
egorical_accuracy: 0.8832 - precision: 0.9069 - val_loss: 0.3395 - val_categor
ical_accuracy: 0.8788 - val_precision: 0.9005
Epoch 10/15
160/160 [==============================] - 20s 125ms/step - loss: 0.3092 - cat
egorical_accuracy: 0.8848 - precision: 0.9070 - val_loss: 0.3370 - val_categor
ical_accuracy: 0.8786 - val_precision: 0.9023
Epoch 11/15
160/160 [==============================] - 20s 125ms/step - loss: 0.3066 - cat
egorical_accuracy: 0.8851 - precision: 0.9076 - val_loss: 0.3350 - val_categor
ical_accuracy: 0.8782 - val_precision: 0.9032
Epoch 12/15
160/160 [==============================] - 20s 125ms/step - loss: 0.2953 - cat
egorical_accuracy: 0.8911 - precision: 0.9135 - val_loss: 0.3280 - val_categor
ical_accuracy: 0.8813 - val_precision: 0.9041
Epoch 13/15
160/160 [==============================] - 20s 124ms/step - loss: 0.2944 - cat
egorical_accuracy: 0.8893 - precision: 0.9087 - val_loss: 0.3300 - val_categor
ical_accuracy: 0.8821 - val_precision: 0.9019
Epoch 14/15
160/160 [==============================] - 20s 125ms/step - loss: 0.2865 - cat
egorical_accuracy: 0.8944 - precision: 0.9138 - val_loss: 0.3437 - val_categor
ical_accuracy: 0.8774 - val_precision: 0.8992
Epoch 15/15
160/160 [==============================] - 20s 124ms/step - loss: 0.2908 - cat
egorical_accuracy: 0.8932 - precision: 0.9122 - val_loss: 0.3287 - val_categor
ical_accuracy: 0.8808 - val_precision: 0.9031
INFO:tensorflow:Assets written to: dropfcnn/dropfcnn.1/assets
Model: "sequential_15"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_46 (Dense)             (None, 28, 28, 64)        128       
_________________________________________________________________
flatten_15 (Flatten)         (None, 50176)             0         
_________________________________________________________________
dense_47 (Dense)             (None, 64)                3211328   
_________________________________________________________________
dropout_5 (Dropout)          (None, 64)                0         
_________________________________________________________________
dense_48 (Dense)             (None, 64)                4160      
_________________________________________________________________
dense_49 (Dense)             (None, 64)                4160      
_________________________________________________________________
dense_50 (Dense)             (None, 10)                650       
=================================================================
Total params: 3,220,426
Trainable params: 3,220,426
Non-trainable params: 0
_________________________________________________________________
Epoch 1/15
160/160 [==============================] - 44s 275ms/step - loss: 1.2853 - cat
egorical_accuracy: 0.6801 - precision: 0.8315 - val_loss: 0.5809 - val_categor
ical_accuracy: 0.8097 - val_precision: 0.9050
Epoch 2/15
160/160 [==============================] - 30s 187ms/step - loss: 0.7406 - cat
egorical_accuracy: 0.7132 - precision: 0.8348 - val_loss: 0.6438 - val_categor
ical_accuracy: 0.7479 - val_precision: 0.8757
Epoch 3/15
160/160 [==============================] - 24s 151ms/step - loss: 0.6763 - cat
egorical_accuracy: 0.7400 - precision: 0.8497 - val_loss: 0.5755 - val_categor
ical_accuracy: 0.7911 - val_precision: 0.8891
Epoch 4/15
160/160 [==============================] - 12s 74ms/step - loss: 0.6364 - cate
gorical_accuracy: 0.7587 - precision: 0.8583 - val_loss: 0.5632 - val_categori
cal_accuracy: 0.7927 - val_precision: 0.9005
Epoch 5/15
160/160 [==============================] - 12s 75ms/step - loss: 0.6183 - cate
gorical_accuracy: 0.7652 - precision: 0.8602 - val_loss: 0.6849 - val_categori
```

```
cal_accuracy: 0.7120 - val_precision: 0.7889
INFO:tensorflow:Assets written to: dropfcnn/dropfcnn.2/assets
Model: "sequential_16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_51 (Dense)             (None, 28, 28, 64)        128
_________________________________________________________________
flatten_16 (Flatten)         (None, 50176)             0
_________________________________________________________________
dense_52 (Dense)             (None, 64)                3211328
_________________________________________________________________
dense_53 (Dense)             (None, 64)                4160
_________________________________________________________________
dropout_6 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_54 (Dense)             (None, 64)                4160
_________________________________________________________________
dense_55 (Dense)             (None, 10)                650
=================================================================
Total params: 3,220,426
Trainable params: 3,220,426
Non-trainable params: 0
_________________________________________________________________
Epoch 1/15
160/160 [==============================] - 20s 123ms/step - loss: 1.2121 - cat
egorical_accuracy: 0.6514 - precision: 0.7863 - val_loss: 0.5142 - val_categor
ical_accuracy: 0.8048 - val_precision: 0.8792
Epoch 2/15
160/160 [==============================] - 14s 87ms/step - loss: 0.5746 - cate
gorical_accuracy: 0.7947 - precision: 0.8662 - val_loss: 0.4500 - val_categori
cal_accuracy: 0.8295 - val_precision: 0.8891
Epoch 3/15
160/160 [==============================] - 18s 113ms/step - loss: 0.4982 - cat
egorical_accuracy: 0.8230 - precision: 0.8797 - val_loss: 0.4257 - val_categor
ical_accuracy: 0.8462 - val_precision: 0.8983
Epoch 4/15
160/160 [==============================] - 16s 101ms/step - loss: 0.4662 - cat
egorical_accuracy: 0.8343 - precision: 0.8852 - val_loss: 0.4188 - val_categor
ical_accuracy: 0.8443 - val_precision: 0.8821
Epoch 5/15
160/160 [==============================] - 12s 75ms/step - loss: 0.4344 - cate
gorical_accuracy: 0.8480 - precision: 0.8953 - val_loss: 0.4101 - val_categori
cal_accuracy: 0.8478 - val_precision: 0.8978
Epoch 6/15
160/160 [==============================] - 12s 76ms/step - loss: 0.4163 - cate
gorical_accuracy: 0.8523 - precision: 0.8988 - val_loss: 0.4009 - val_categori
cal_accuracy: 0.8546 - val_precision: 0.9017
Epoch 7/15
160/160 [==============================] - 12s 74ms/step - loss: 0.3968 - cate
gorical_accuracy: 0.8593 - precision: 0.8999 - val_loss: 0.3974 - val_categori
cal_accuracy: 0.8514 - val_precision: 0.8921
Epoch 8/15
160/160 [==============================] - 12s 74ms/step - loss: 0.3870 - cate
gorical_accuracy: 0.8625 - precision: 0.9010 - val_loss: 0.3850 - val_categori
cal_accuracy: 0.8617 - val_precision: 0.9022
Epoch 9/15
160/160 [==============================] - 12s 75ms/step - loss: 0.3796 - cate
gorical_accuracy: 0.8652 - precision: 0.9016 - val_loss: 0.3779 - val_categori
cal_accuracy: 0.8637 - val_precision: 0.8964
Epoch 10/15
160/160 [==============================] - 12s 76ms/step - loss: 0.3651 - cate
gorical_accuracy: 0.8688 - precision: 0.9051 - val_loss: 0.3992 - val_categori
cal_accuracy: 0.8633 - val_precision: 0.8967
Epoch 11/15
160/160 [==============================] - 12s 73ms/step - loss: 0.3546 - cate
gorical_accuracy: 0.8717 - precision: 0.9057 - val_loss: 0.3802 - val_categori
cal_accuracy: 0.8663 - val_precision: 0.8962
Epoch 12/15
```

```
160/160 [==============================] - 12s 72ms/step - loss: 0.3524 - cate
gorical_accuracy: 0.8723 - precision: 0.9061 - val_loss: 0.3832 - val_categori
cal_accuracy: 0.8623 - val_precision: 0.8996
Epoch 13/15
160/160 [==============================] - 12s 72ms/step - loss: 0.3454 - cate
gorical_accuracy: 0.8745 - precision: 0.9072 - val_loss: 0.3703 - val_categori
cal_accuracy: 0.8668 - val_precision: 0.9042
Epoch 14/15
160/160 [==============================] - 12s 72ms/step - loss: 0.3391 - cate
gorical_accuracy: 0.8799 - precision: 0.9113 - val_loss: 0.3803 - val_categori
cal_accuracy: 0.8634 - val_precision: 0.8911
Epoch 15/15
160/160 [==============================] - 11s 72ms/step - loss: 0.3350 - cate
gorical_accuracy: 0.8792 - precision: 0.9111 - val_loss: 0.3827 - val_categori
cal_accuracy: 0.8633 - val_precision: 0.8974
INFO:tensorflow:Assets written to: dropfcnn/dropfcnn.3/assets
Model: "sequential_17"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_56 (Dense)             (None, 28, 28, 64)        128       
_________________________________________________________________
flatten_17 (Flatten)         (None, 50176)             0         
_________________________________________________________________
dense_57 (Dense)             (None, 64)                3211328   
_________________________________________________________________
dense_58 (Dense)             (None, 64)                4160      
_________________________________________________________________
dense_59 (Dense)             (None, 64)                4160      
_________________________________________________________________
dropout_7 (Dropout)          (None, 64)                0         
_________________________________________________________________
dense_60 (Dense)             (None, 10)                650       
=================================================================
Total params: 3,220,426
Trainable params: 3,220,426
Non-trainable params: 0
_________________________________________________________________
Epoch 1/15
160/160 [==============================] - 26s 161ms/step - loss: 1.1212 - cat
egorical_accuracy: 0.7404 - precision: 0.8591 - val_loss: 0.4928 - val_categor
ical_accuracy: 0.8207 - val_precision: 0.8837
Epoch 2/15
160/160 [==============================] - 14s 88ms/step - loss: 0.5054 - cate
gorical_accuracy: 0.8286 - precision: 0.8812 - val_loss: 0.4221 - val_categori
cal_accuracy: 0.8456 - val_precision: 0.8854
Epoch 3/15
160/160 [==============================] - 11s 71ms/step - loss: 0.4374 - cate
gorical_accuracy: 0.8470 - precision: 0.8910 - val_loss: 0.3946 - val_categori
cal_accuracy: 0.8533 - val_precision: 0.8865
Epoch 4/15
160/160 [==============================] - 12s 72ms/step - loss: 0.4083 - cate
gorical_accuracy: 0.8585 - precision: 0.8982 - val_loss: 0.3795 - val_categori
cal_accuracy: 0.8607 - val_precision: 0.8925
Epoch 5/15
160/160 [==============================] - 12s 74ms/step - loss: 0.3856 - cate
gorical_accuracy: 0.8627 - precision: 0.8997 - val_loss: 0.3713 - val_categori
cal_accuracy: 0.8634 - val_precision: 0.8992
Epoch 6/15
160/160 [==============================] - 12s 74ms/step - loss: 0.3679 - cate
gorical_accuracy: 0.8692 - precision: 0.9044 - val_loss: 0.3760 - val_categori
cal_accuracy: 0.8637 - val_precision: 0.8950
Epoch 7/15
160/160 [==============================] - 12s 77ms/step - loss: 0.3635 - cate
gorical_accuracy: 0.8710 - precision: 0.9065 - val_loss: 0.3602 - val_categori
cal_accuracy: 0.8720 - val_precision: 0.9039
Epoch 8/15
160/160 [==============================] - 13s 78ms/step - loss: 0.3476 - cate
gorical_accuracy: 0.8753 - precision: 0.9093 - val_loss: 0.3680 - val_categori
```

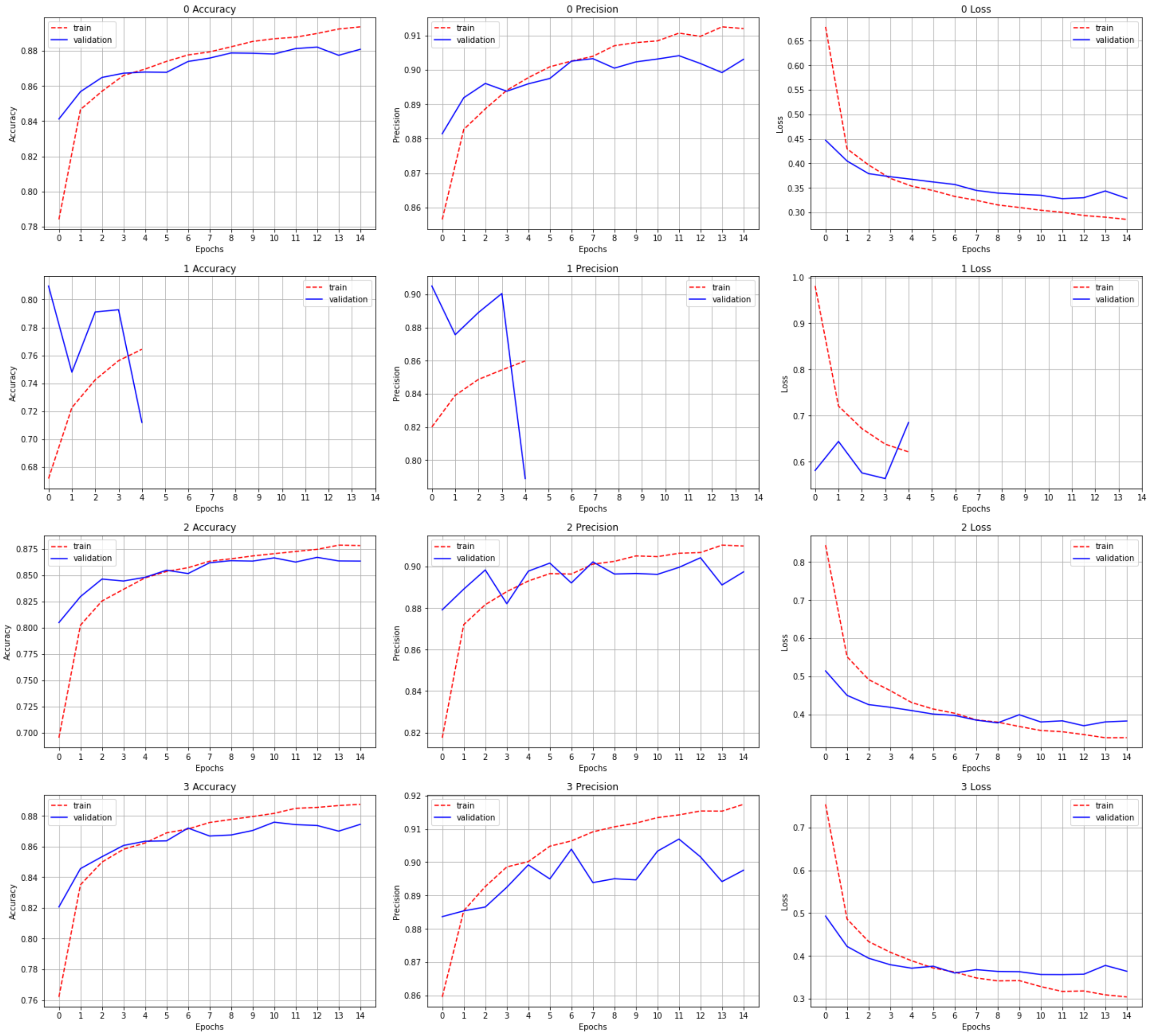
```
cal_accuracy: 0.8668 - val_precision: 0.8939
Epoch 9/15
160/160 [==============================] - 12s 74ms/step - loss: 0.3349 - cate
gorical_accuracy: 0.8806 - precision: 0.9130 - val_loss: 0.3638 - val_categori
cal_accuracy: 0.8675 - val_precision: 0.8950
Epoch 10/15
160/160 [==============================] - 12s 73ms/step - loss: 0.3439 - cate
gorical_accuracy: 0.8804 - precision: 0.9118 - val_loss: 0.3632 - val_categori
cal_accuracy: 0.8704 - val_precision: 0.8947
Epoch 11/15
160/160 [==============================] - 12s 73ms/step - loss: 0.3228 - cate
gorical_accuracy: 0.8829 - precision: 0.9140 - val_loss: 0.3566 - val_categori
cal_accuracy: 0.8758 - val_precision: 0.9033
Epoch 12/15
160/160 [==============================] - 12s 77ms/step - loss: 0.3166 - cate
gorical_accuracy: 0.8852 - precision: 0.9146 - val_loss: 0.3564 - val_categori
cal_accuracy: 0.8742 - val_precision: 0.9069
Epoch 13/15
160/160 [==============================] - 13s 79ms/step - loss: 0.3166 - cate
gorical_accuracy: 0.8859 - precision: 0.9162 - val_loss: 0.3573 - val_categori
cal_accuracy: 0.8737 - val_precision: 0.9016
Epoch 14/15
160/160 [==============================] - 13s 78ms/step - loss: 0.3063 - cate
gorical_accuracy: 0.8879 - precision: 0.9168 - val_loss: 0.3778 - val_categori
cal_accuracy: 0.8700 - val_precision: 0.8942
Epoch 15/15
160/160 [==============================] - 12s 76ms/step - loss: 0.3014 - cate
gorical_accuracy: 0.8890 - precision: 0.9180 - val_loss: 0.3642 - val_categori
cal_accuracy: 0.8744 - val_precision: 0.8976
INFO:tensorflow:Assets written to: dropfcnn/dropfcnn.4/assets
```

FCNN refactored performance

As you can see from the below graphs, once the model has the dropout layer applied it is able to score higher for accuracy and precision on the validation stage consistently through out each of the models, whilst also keeping a low loss rating.

In [22]:

```
visualise(dropfcnn_count, dropfcnn_history)
```



Parameter tuning

When creating a machine learning model it is near impossible to find the perfect parameters without using a parameter tuning method. These work by you defining a grid of parameters with an array of values. The tuner then uses 1 parameter from each array in the grid and applies it to the appropriate parameter in your model. The model is then trained using those selected parameters. Whilst the tuner is training the model it is either trying to maximize or minimize an objective function. The result of this objective function leads the tuner in the way of the best hyperparameters for the model.

Here the following parameters are being tuned for the CNN model; number of filters in the convolutional layers, number of units in the fully connected layer, the rate of dropout in the dropout layer and finally the learning rate for the adam optimiser. For the FCNN model the parameters tuned are; number of units in the fully connected layers, the rate of dropout in the dropout layer and the learning rate for the adam optimiser.

The parametertuning.py script has been made seperate from this notebook, because the total run time of it exceeds 17 hours.

After finding the optimal parameters by runing the parametertuning.py script, the models need to be created and have those parameters added in the correct places.

In [24]:

```
cnndf = pd.read_csv('results/cnn')
# CNN parameters
cnn_filters1 = int(cnndf['first'].values)
cnn_filters2 = int(cnndf['second'].values)
cnn_units = int(cnndf['connected'].values)
cnn_dropout = float(cnndf['dropout'].values)
cnn_learning_rate = float(cnndf['learning_rate'].values)
cnn_optimizer = Adam(cnn_learning_rate)
cnndf
```

Out[24]:

	Unnamed: 0	first	second	connected	dropout	learning_rate
0	parameters	256	256	256	0.5	0.001

In [25]:

```
cnn = Sequential()
cnn.add(Input(INPUT_SHAPE))
cnn.add(Conv2D(cnn_filters1, kernel_size=KERNEL_SIZE ,activation=ACTIV[0], pa
cnn.add(MaxPool2D(POOL_SIZE))
cnn.add(Conv2D(cnn_filters2, kernel_size=KERNEL_SIZE ,activation=ACTIV[0], pa
cnn.add(MaxPool2D(POOL_SIZE))
cnn.add(Dropout(cnn_dropout))
cnn.add(Flatten())
cnn.add(Dense(cnn_units, activation=ACTIV[0]))
cnn.add(Dense(NUM_CLASSES, activation=ACTIV[1]))
cnn.compile(loss=LOSS, optimizer=cnn_optimizer, metrics=METRICS)
cnn.summary()
```

```
Model: "sequential_18"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_20 (Conv2D)           (None, 28, 28, 256)       6656      
_________________________________________________________________
max_pooling2d_20 (MaxPooling (None, 14, 14, 256)       0         
_________________________________________________________________
conv2d_21 (Conv2D)           (None, 14, 14, 256)       1638656   
_________________________________________________________________
max_pooling2d_21 (MaxPooling (None, 7, 7, 256)         0         
_________________________________________________________________
dropout_8 (Dropout)          (None, 7, 7, 256)         0         
_________________________________________________________________
flatten_18 (Flatten)         (None, 12544)             0         
_________________________________________________________________
dense_61 (Dense)             (None, 256)               3211520   
_________________________________________________________________
dense_62 (Dense)             (None, 10)                2570      
=================================================================
Total params: 4,859,402
Trainable params: 4,859,402
Non-trainable params: 0
_________________________________________________________________
```

In [26]:

```
fcnndf = pd.read_csv('results/fcnn')
# FCNN parameters
fcnn_units1 = int(fcnndf['first'].values)
fcnn_units2 = int(fcnndf['second'].values)
fcnn_units3 = int(fcnndf['third'].values)
fcnn_units4 = int(fcnndf['fourth'].values)
fcnn_dropout = float(fcnndf['dropout'].values)
fcnn_learning_rate = float(fcnndf['learning_rate'].values)
fcnn_optimizer = Adam(fcnn_learning_rate)
fcnndf
```

Out[26]:

	Unnamed: 0	first	second	third	fourth	dropout	learning_rate
0	parameters	32	256	256	256	0.6	0.0001

In [27]:

```
fcnn = Sequential()
fcnn.add(Input(INPUT_SHAPE))
fcnn.add(Dense(fcnn_units1, activation=ACTIV[0]))
fcnn.add(Flatten())
fcnn.add(Dense(fcnn_units2, activation=ACTIV[0]))
fcnn.add(Dense(fcnn_units3, activation=ACTIV[0]))
fcnn.add(Dense(fcnn_units4, activation=ACTIV[0]))
fcnn.add(Dropout(fcnn_dropout))
fcnn.add(Dense(NUM_CLASSES, activation=ACTIV[1]))
fcnn.compile(loss=LOSS, optimizer=fcnn_optimizer, metrics=METRICS)
```

Once the parameters have been added to the models, the models need to be trained on the training data before they can be evaluated

In [28]:

```
cnn_history = cnn.fit(x_train, y_train, BATCH_SIZE, EPOCHS, validation_split=
```

```
Epoch 1/15
160/160 [==============================] - 408s 3s/step - loss: 0.7522 - categ
orical_accuracy: 0.8058 - precision: 0.8742 - val_loss: 0.3227 - val_categoric
al_accuracy: 0.8863 - val_precision: 0.9116
Epoch 2/15
160/160 [==============================] - 409s 3s/step - loss: 0.3225 - categ
orical_accuracy: 0.8830 - precision: 0.9057 - val_loss: 0.2722 - val_categoric
al_accuracy: 0.9013 - val_precision: 0.9188
Epoch 3/15
160/160 [==============================] - 396s 2s/step - loss: 0.2672 - categ
orical_accuracy: 0.9022 - precision: 0.9179 - val_loss: 0.2507 - val_categoric
al_accuracy: 0.9082 - val_precision: 0.9240
Epoch 4/15
160/160 [==============================] - 391s 2s/step - loss: 0.2262 - categ
orical_accuracy: 0.9154 - precision: 0.9279 - val_loss: 0.2453 - val_categoric
al_accuracy: 0.9093 - val_precision: 0.9228
Epoch 5/15
160/160 [==============================] - 377s 2s/step - loss: 0.2080 - categ
orical_accuracy: 0.9213 - precision: 0.9329 - val_loss: 0.2346 - val_categoric
al_accuracy: 0.9152 - val_precision: 0.9235
Epoch 6/15
160/160 [==============================] - 376s 2s/step - loss: 0.1908 - categ
orical_accuracy: 0.9302 - precision: 0.9391 - val_loss: 0.2261 - val_categoric
al_accuracy: 0.9190 - val_precision: 0.9274
Epoch 7/15
160/160 [==============================] - 412s 3s/step - loss: 0.1724 - categ
orical_accuracy: 0.9355 - precision: 0.9439 - val_loss: 0.2127 - val_categoric
al_accuracy: 0.9218 - val_precision: 0.9312
Epoch 8/15
160/160 [==============================] - 384s 2s/step - loss: 0.1583 - categ
orical_accuracy: 0.9397 - precision: 0.9468 - val_loss: 0.2246 - val_categoric
al_accuracy: 0.9191 - val_precision: 0.9275
Epoch 9/15
160/160 [==============================] - 408s 3s/step - loss: 0.1448 - categ
orical_accuracy: 0.9467 - precision: 0.9524 - val_loss: 0.2198 - val_categoric
al_accuracy: 0.9241 - val_precision: 0.9308
Epoch 10/15
160/160 [==============================] - 402s 3s/step - loss: 0.1304 - categ
orical_accuracy: 0.9514 - precision: 0.9564 - val_loss: 0.2090 - val_categoric
al_accuracy: 0.9235 - val_precision: 0.9323
Epoch 11/15
160/160 [==============================] - 401s 3s/step - loss: 0.1184 - categ
orical_accuracy: 0.9557 - precision: 0.9599 - val_loss: 0.2133 - val_categoric
al_accuracy: 0.9275 - val_precision: 0.9336
```

```
Epoch 12/15
160/160 [==============================] - 415s 3s/step - loss: 0.1053 - categ
orical_accuracy: 0.9592 - precision: 0.9630 - val_loss: 0.2133 - val_categoric
al_accuracy: 0.9295 - val_precision: 0.9358
Epoch 13/15
160/160 [==============================] - 403s 3s/step - loss: 0.0949 - categ
orical_accuracy: 0.9636 - precision: 0.9666 - val_loss: 0.2312 - val_categoric
al_accuracy: 0.9218 - val_precision: 0.9269
Epoch 14/15
160/160 [==============================] - 401s 3s/step - loss: 0.0919 - categ
orical_accuracy: 0.9651 - precision: 0.9674 - val_loss: 0.2346 - val_categoric
al_accuracy: 0.9276 - val_precision: 0.9322
Epoch 15/15
160/160 [==============================] - 382s 2s/step - loss: 0.0795 - categ
orical_accuracy: 0.9706 - precision: 0.9723 - val_loss: 0.2317 - val_categoric
al_accuracy: 0.9293 - val_precision: 0.9343
```

In [29]:

```
fcnn_history = fcnn.fit(x_train, y_train, BATCH_SIZE, EPOCHS, validation_spli
```

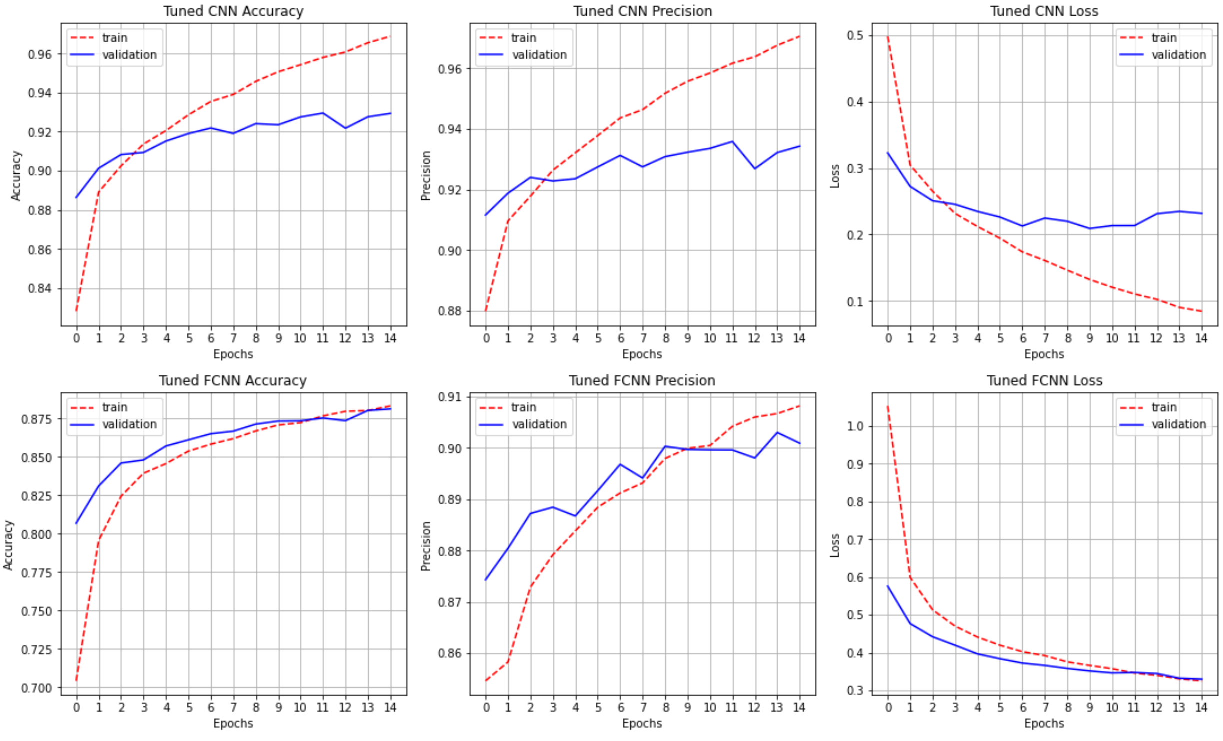
```
Epoch 1/15
160/160 [==============================] - 12s 74ms/step - loss: 1.4412 - cate
gorical_accuracy: 0.7065 - precision: 0.8869 - val_loss: 0.5757 - val_categori
cal_accuracy: 0.8069 - val_precision: 0.8743
Epoch 2/15
160/160 [==============================] - 12s 73ms/step - loss: 0.6299 - cate
gorical_accuracy: 0.7862 - precision: 0.8558 - val_loss: 0.4765 - val_categori
cal_accuracy: 0.8311 - val_precision: 0.8804
Epoch 3/15
160/160 [==============================] - 11s 71ms/step - loss: 0.5301 - cate
gorical_accuracy: 0.8187 - precision: 0.8687 - val_loss: 0.4417 - val_categori
cal_accuracy: 0.8461 - val_precision: 0.8872
Epoch 4/15
160/160 [==============================] - 12s 74ms/step - loss: 0.4700 - cate
gorical_accuracy: 0.8401 - precision: 0.8807 - val_loss: 0.4194 - val_categori
cal_accuracy: 0.8481 - val_precision: 0.8884
Epoch 5/15
160/160 [==============================] - 12s 72ms/step - loss: 0.4461 - cate
gorical_accuracy: 0.8439 - precision: 0.8827 - val_loss: 0.3963 - val_categori
cal_accuracy: 0.8572 - val_precision: 0.8867
Epoch 6/15
160/160 [==============================] - 12s 74ms/step - loss: 0.4176 - cate
gorical_accuracy: 0.8545 - precision: 0.8882 - val_loss: 0.3835 - val_categori
cal_accuracy: 0.8612 - val_precision: 0.8917
Epoch 7/15
160/160 [==============================] - 12s 72ms/step - loss: 0.4066 - cate
gorical_accuracy: 0.8565 - precision: 0.8900 - val_loss: 0.3720 - val_categori
cal_accuracy: 0.8652 - val_precision: 0.8968
Epoch 8/15
160/160 [==============================] - 12s 72ms/step - loss: 0.3937 - cate
gorical_accuracy: 0.8634 - precision: 0.8945 - val_loss: 0.3658 - val_categori
cal_accuracy: 0.8668 - val_precision: 0.8941
Epoch 9/15
160/160 [==============================] - 12s 73ms/step - loss: 0.3768 - cate
gorical_accuracy: 0.8655 - precision: 0.8967 - val_loss: 0.3577 - val_categori
cal_accuracy: 0.8714 - val_precision: 0.9003
Epoch 10/15
160/160 [==============================] - 12s 74ms/step - loss: 0.3719 - cate
gorical_accuracy: 0.8685 - precision: 0.8975 - val_loss: 0.3512 - val_categori
cal_accuracy: 0.8734 - val_precision: 0.8997
Epoch 11/15
160/160 [==============================] - 12s 74ms/step - loss: 0.3541 - cate
gorical_accuracy: 0.8744 - precision: 0.9026 - val_loss: 0.3461 - val_categori
cal_accuracy: 0.8736 - val_precision: 0.8996
Epoch 12/15
160/160 [==============================] - 12s 73ms/step - loss: 0.3401 - cate
gorical_accuracy: 0.8786 - precision: 0.9050 - val_loss: 0.3471 - val_categori
cal_accuracy: 0.8754 - val_precision: 0.8996
Epoch 13/15
```

```
160/160 [==============================] - 12s 74ms/step - loss: 0.3398 - cate
gorical_accuracy: 0.8789 - precision: 0.9047 - val_loss: 0.3443 - val_categori
cal_accuracy: 0.8737 - val_precision: 0.8980
Epoch 14/15
160/160 [==============================] - 12s 74ms/step - loss: 0.3286 - cate
gorical_accuracy: 0.8802 - precision: 0.9078 - val_loss: 0.3317 - val_categori
cal_accuracy: 0.8803 - val_precision: 0.9030
Epoch 15/15
160/160 [==============================] - 12s 73ms/step - loss: 0.3239 - cate
gorical_accuracy: 0.8842 - precision: 0.9079 - val_loss: 0.3294 - val_categori
cal_accuracy: 0.8813 - val_precision: 0.9009
```

Now their training performances can be compared

In [30]:

```python
historys = [cnn_history, fcnn_history]
names = ['Tuned CNN', 'Tuned FCNN']
fig, axs = plt.subplots(2, 3, figsize=(15, 9), constrained_layout=True)

metrics = [('categorical_accuracy', 'val_categorical_accuracy'), ('precision'
titles = ['Accuracy', 'Precision', 'Loss']

for idx, ax in enumerate(axs):
    for i in range(3):
        ax[i].plot(historys[idx].history[metrics[i][0]], label='train', lines
        ax[i].plot(historys[idx].history[metrics[i][1]], label='validation', 
        ax[i].set_title((f'{names[idx]} {titles[i]}'))
        ax[i].set_ylabel(titles[i])
        ax[i].set_xlabel('Epochs')
        ax[i].set_xticks([x for x in np.arange(15)])
        ax[i].legend()
        ax[i].grid()
```



Testing

Now that the models have been trainined with their optimal hyperparameters, we can move onto evaluating the models performance on previously unseen data (Testing samples).

In [31]:

```python
model_scores = []
models = [cnn, fcnn]
```

```
for i in np.arange(len(models)):
    scores = models[i].evaluate(x_test, y_test)
    model_scores.append(scores)
```

```
313/313 [==============================] - 22s 70ms/step - loss: 0.2533 - cate
gorical_accuracy: 0.9248 - precision: 0.9292
313/313 [==============================] - 2s 4ms/step - loss: 0.3581 - catego
rical_accuracy: 0.8724 - precision: 0.8932
```

In [48]:

```
df = pd.DataFrame(model_scores, index=['CNN', 'FCNN'], columns=['Loss', 'Accu
df.style.format('{:.4f}').highlight_min('Loss', color='lightgreen').highlight_
```

Out[48]:

	Loss	Accuracy	Precision
CNN	0.2533	0.9248	0.9292
FCNN	0.3581	0.8724	0.8932

Other works in literature

The following table and graph compares the test accuracy of my models against others that I have found in literature. (Xiao, Rasul and Vollgraf, 2017) - SVC, (Bhatnagar, Ghosal and Kolekar, 2017) - CNN2, BATCH, SKIP, (Agarap, 2017) - CNNSO, CNNSV, (Dufourq and Bassett, 2017) - EDEN, (Greeshma and Sreekumar, 2019) - HPO

In [47]:

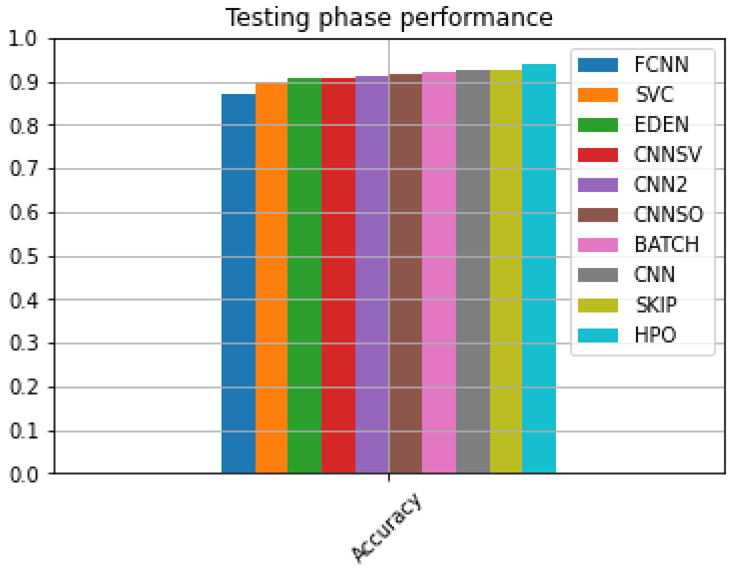
```
compared_scores = [[model_scores[0][1], model_scores[1][1], 0.8970, 0.9060, 0
names = ['CNN', 'FCNN', 'SVC', 'EDEN', 'CNN2', 'BATCH', 'SKIP', 'CNNSO', 'CNN

compared_df = pd.DataFrame(compared_scores, columns=names, index=['Accuracy']
compared_df.sort_values(by='Accuracy', axis=1, inplace=True)
compared_df.style.format('{:.4f}')
```

Out[47]:

	FCNN	SVC	EDEN	CNNSV	CNN2	CNNSO	BATCH	CNN	SKIP	HPO
Accuracy	0.8724	0.8970	0.9060	0.9072	0.9117	0.9186	0.9222	0.9248	0.9254	0.9399

In [40]:

```
compared_df.plot.bar()
plt.xticks(rotation=45)
plt.title('Testing phase performance')
plt.yticks([x for x in np.arange(0.0, 100.0, 10.0)])
plt.grid()
```



Conclusion

The CNN model was able to achieve much better scores for accuracy and loss than the FCNN model and only a slight increase in the precision. However the final model, even with the tuned parameters was still overfitting on the training data. This is something that needs to be addressed prior to employing the model into real world scenarios.

Now if you compare the accuracy of my models to the other models in literature, you can see that the CNN model fairs very well against the other CNN models that have applied either different techniques for countering overfitting, different parameters, different architectures and finally different types of data preprocessing steps. On the other hand, the FCNN model is lacking behind the pack.

With this being said the training time my final CNN model took per epoch was on average around 380 seconds whereas the final FCNN model took around 12 seconds per epoch. So even though the CNN model is able to outperform the FCNN model on the performance metrics of accuracy and precision, the computational cost of performing convolutions is too great to not apply a mechanism to reduce the time taken to train the model. As presented in (Greeshma and Sreekumar, 2019) paper.

Applications of CNNs in the real world

1. 'Object detection: With CNN, we now have sophisticated models like R-CNN, Fast R-CNN and Faster R-CNN that are the predominant pipline for many object detction models deployed in autonomous vehicles, facial detection, and more.' (Mishra, 2020)
2. 'Semantic segmentation: In 2015, a group of researchers from Hong Kong developed a CNN-based Deep Parsing Network to incorporate rich information into an image sementation model. researches from UC Berkeley alo built fully convolutional networks taht improved upon state-of-the-art semantic segmentation.' (Mishra, 2020)
3. 'Image captioning: CNNs are used with recurrent neural networks to write captions for images and videos. This can be used for many applications such as activity recognition or describing videos and images for the visually impaired. It has been heavily deployed by YouTube to make sense to the huge number of vidoes uploaded to the platform on a regulard basis.' (Mishra, 2020)

References

Agarap, A.F.M (2017). An Architecture Combining Convolutional Neural Network (CNN) and Support Vector Machine (SVM) for Image Classification. [online] Available at: https://arxiv.org/pdf/1712.03541.pdf [Accessed 6 Dec. 2021].

Bhatnagar, S., Ghosal, D. and Kolekar, M.H. (2017). Classification of Fashion Article Images Using Convolutional Neural Networks. [online] Ieeexplore. Available at: https://ieeexplore.ieee.org/abstract/document/8313740 [Accessed 15 Jan. 2022].

Brownlee, J. (2021). Code Adam Optimization Algorithm From Scratch. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/adam-optimization-from-scratch/ [Accessed 15 Dec. 2021].

Dufourq, E. and Bassett, B.A. (2017). EDEN: Evolutionary deep networks for efficient machine learning. [online] IEEE Xplore. Available at: https://ieeexplore.ieee.org/abstract/document/8261132 [Accessed 7 Jan. 2022].

Dürr, O., Stick, B. and Murina, E. (2020). Chapter 2: Neural network architectures · Probabilistic Deep Learning: With Python, Keras and TensorFlow Probability. [online] livebook.manning.com. Available at: https://livebook.manning.com/book/probabilistic-deep-learning-with-python/chapter-2/v-5/10 [Accessed 17 Jan. 2022].

Greeshma, K.V. and Sreekumar, K. (2019). Hyperparameter Optimization and Regularization on Fashion-MNIST Classification. [online] Research Gate. Available at: https://www.researchgate.net/profile/Greeshma-K-V/publication/334947180_Hyperparameter_Optimization_and_Regularization_on_Fashion-MNIST_Classification/links/5d45c99a4585153e593ae361/Hyperparameter-Optimization-and-Regularization-on-Fashion-MNIST-Classification.pdf [Accessed 12 Dec. 2021].

Gupta, S. (2018). Understanding Image Recognition and Its Uses. [online] Einfochips. Available at: https://www.einfochips.com/blog/understanding-image-recognition-and-its-uses [Accessed 1 Dec. 2021].

IBM Cloud Education (2020). What are Neural Networks? [online] www.ibm.com. Available at: https://www.ibm.com/uk-en/cloud/learn/neural-networks [Accessed 20 Dec. 2021].

Kayed, M., Anter, A. and Mohamed, H. (2020). Classification of Garments from Fashion MNIST Dataset Using CNN LeNet-5 Architecture. [online] IEEE Xplore. Available at: https://ieeexplore.ieee.org/document/9047776 [Accessed 7 Dec. 2021].

Mishra, M. (2020). Convolutional Neural Networks, Explained. [online] Medium. Available at: https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939 [Accessed 19 Jan. 2022].

Moore, C.M. (2019). Fully Connected Neural Network | Radiology Reference Article | Radiopaedia.org. [online] Radiopaedia. Available at: https://radiopaedia.org/articles/fully-connected-neural-network [Accessed 2 Jan. 2022].

Saha, S. (2018). A Comprehensive Guide to Convolutional Neural Networks—the ELI5 way. [online] Towards Data Science. Available at: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 [Accessed 30 Dec. 2021].

Xiao, H., Rasul, K. and Vollgraf, R. (2017). Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. [online] Available at: https://arxiv.org/pdf/1708.07747.pdf [Accessed 5 Dec. 2021].

In [ ]:

```

```