Zulema Perez

CPSC 585

Project 7: Convolutional Neural Networks

In [ ]:

```
import sys

from google.colab import drive

drive.mount('/content/drive')
sys.path.insert(0,'/content/drive/MyDrive/Colab Notebooks/')
```

```
Mounted at /content/drive
```

In [ ]:

```
import numpy as np

#from google.colab import files
#data_to_load = files.upload()

path = '/content/drive/MyDrive/emnist_letters.npz'

num_classes = 27
input_shape = (28, 28, 1)

# Load EMINST data
with np.load(path, allow_pickle=True) as data:
  x_train = data['train_images']
  y_train = data['train_labels']
  x_test = data['test_images']
  y_test = data['test_labels']
  x_validate = data['validate_images']
  y_validate = data['validate_labels']

  x_train = x_train.reshape(x_train.shape[0], 28, 28)
  x_test = x_test.reshape(x_test.shape[0], 28, 28)
  x_validate = x_validate.reshape(x_validate.shape[0], 28, 28)

  x_train = x_train.astype("float32")/255
  x_test = x_test.astype("float32")/255
  x_validate = x_validate.astype("float32")/255

  x_train = np.expand_dims(x_train, -1)
  x_test = np.expand_dims(x_test, -1)
  x_validate = np.expand_dims(x_validate, -1)
```

In [ ]:

```
print(x_train.shape)
```

```
(104000, 28, 28, 1)
```

In [ ]:

```
print(y_train.shape)
print(len(y_train))
```

```
(104000, 27)
```

```
104000
```

In [ ]:

```python
print(y_train)
```

```
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 1. 0.]]
```

In [ ]:

```python
print(x_test.shape)
```

```
(20800, 28, 28, 1)
```

In [ ]:

```python
print(y_test.shape)
print(len(y_test))
```

```
(20800, 27)
20800
```

In [ ]:

```python
y_test
```

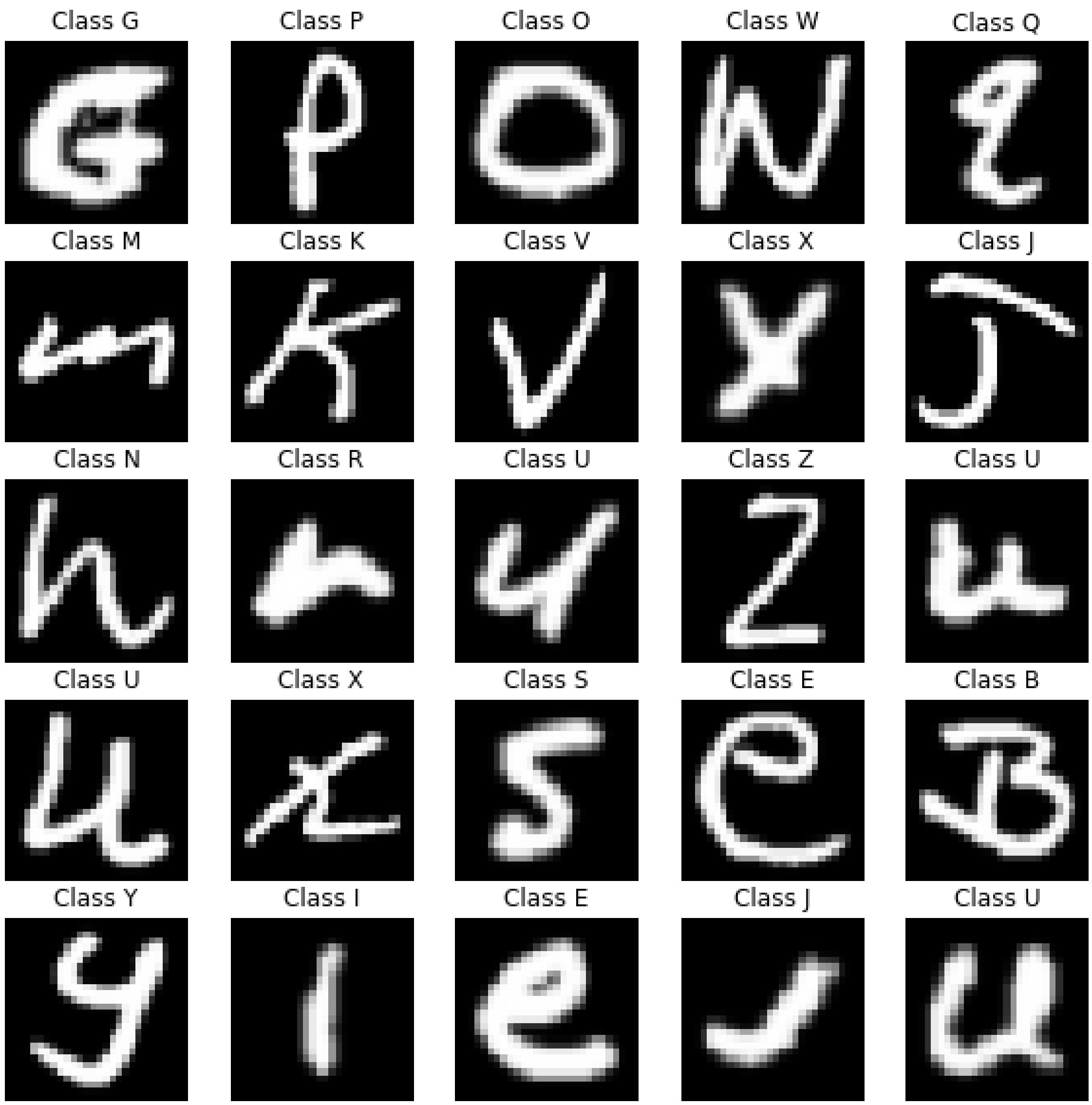
Out[ ]:

```
array([[0., 1., 0., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 1.],
       [0., 0., 0., ..., 0., 0., 1.],
       [0., 0., 0., ..., 0., 0., 1.]], dtype=float32)
```

Experiment 1 Use plt.imshow() to verify that the image data has been loaded correctly and that the corresponding labels are correct.

In [ ]:

```python
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

fig = plt.figure(figsize=(10,10))

cols = 5
rows = 5

for i in range(1, (cols*rows) + 1):
  fig.add_subplot(rows, cols, i)
  plt.axis('off')
  plt.title("Class {}".format(chr(np.argmax(y_train[i])+64)))
  plt.imshow(x_train[i].reshape([28,28]), cmap='Greys_r', interpolation='none')
plt.show()


#for i in range(15):
  #print(chr(np.argmax(y_train[i])+64))
```



Experiment 2

The Keras examples include a Simple MNIST convnet. Note the accuracy obtained by that code compared to Chollet's example from the previous project. Apply this architecture to the EMNIST Letters data.

What accuracy do you achieve? Project 7 - Experiment 2

Test accuracy: 0.8823077082633972

How does this compare with the accuracy for MNIST?

The simple MNIST convet -

Test accuracy of 0.9922000169754028

Much higher than when used on the EMNIST Letters data.

If you completed Project 5, how does this compare with the accuracy you achieved in that project?

Project 5 - Experiment 2

Test Accuracy: 0.9010096192359924

In [ ]:

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras import models
from tensorflow.keras import callbacks
from keras.layers import BatchNormalization
from keras.optimizers import Adam


model = keras.Sequential(
    [
     keras.Input(shape=input_shape),
     layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
     layers.MaxPooling2D(pool_size=(2, 2)),
     layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
     layers.MaxPooling2D(pool_size=(2, 2)),
     layers.Flatten(),
     layers.Dropout(0.5),
     layers.Dense(num_classes, activation="softmax")
    ]
)

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 1600)              0
_________________________________________________________________
dropout (Dropout)            (None, 1600)              0
_________________________________________________________________
dense (Dense)                (None, 27)                43227
=================================================================
Total params: 62,043
Trainable params: 62,043
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

```
batch_size = 128
epochs = 15

model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])
history = model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_
```

```
Epoch 1/15
813/813 [==============================] - 78s 95ms/step - loss: 2.8006 - accuracy: 0.19
33 - val_loss: 1.2642 - val_accuracy: 0.6355
```

```
Epoch 2/15
813/813 [==============================] - 76s 93ms/step - loss: 1.3430 - accuracy: 0.59
83 - val_loss: 1.0088 - val_accuracy: 0.7079
Epoch 3/15
813/813 [==============================] - 76s 93ms/step - loss: 1.1105 - accuracy: 0.66
51 - val_loss: 0.7936 - val_accuracy: 0.7663
Epoch 4/15
813/813 [==============================] - 76s 94ms/step - loss: 0.9096 - accuracy: 0.72
21 - val_loss: 0.6569 - val_accuracy: 0.8060
Epoch 5/15
813/813 [==============================] - 76s 93ms/step - loss: 0.7860 - accuracy: 0.76
32 - val_loss: 0.5760 - val_accuracy: 0.8283
Epoch 6/15
813/813 [==============================] - 76s 94ms/step - loss: 0.7060 - accuracy: 0.78
57 - val_loss: 0.5314 - val_accuracy: 0.8428
Epoch 7/15
813/813 [==============================] - 79s 97ms/step - loss: 0.6578 - accuracy: 0.79
76 - val_loss: 0.5001 - val_accuracy: 0.8500
Epoch 8/15
813/813 [==============================] - 76s 93ms/step - loss: 0.6257 - accuracy: 0.80
82 - val_loss: 0.4748 - val_accuracy: 0.8581
Epoch 9/15
813/813 [==============================] - 76s 93ms/step - loss: 0.5918 - accuracy: 0.81
83 - val_loss: 0.4603 - val_accuracy: 0.8623
Epoch 10/15
813/813 [==============================] - 76s 93ms/step - loss: 0.5726 - accuracy: 0.82
39 - val_loss: 0.4435 - val_accuracy: 0.8682
Epoch 11/15
813/813 [==============================] - 76s 94ms/step - loss: 0.5589 - accuracy: 0.82
86 - val_loss: 0.4254 - val_accuracy: 0.8715
Epoch 12/15
813/813 [==============================] - 76s 93ms/step - loss: 0.5428 - accuracy: 0.83
28 - val_loss: 0.4183 - val_accuracy: 0.8737
Epoch 13/15
813/813 [==============================] - 76s 93ms/step - loss: 0.5293 - accuracy: 0.83
62 - val_loss: 0.4072 - val_accuracy: 0.8776
Epoch 14/15
813/813 [==============================] - 76s 93ms/step - loss: 0.5159 - accuracy: 0.84
02 - val_loss: 0.3991 - val_accuracy: 0.8793
Epoch 15/15
813/813 [==============================] - 78s 96ms/step - loss: 0.5106 - accuracy: 0.84
26 - val_loss: 0.3911 - val_accuracy: 0.8817
```

In [ ]:

```python
score = model.evaluate(x_test, y_test, verbose=0)
print("Test loss: ", score[0])
print("Test accuracy: ", score[1])
```

```
Test loss:  0.39253994822502136
Test accuracy:  0.8823077082633972
```

Experiment 3

While the fit() method provides a progress bar and some metrics for each epoch, it is often easier to visualize the training process by plotting a loss curve. Use the History object that this method returns to plot how the loss changes with the number of training epochs.

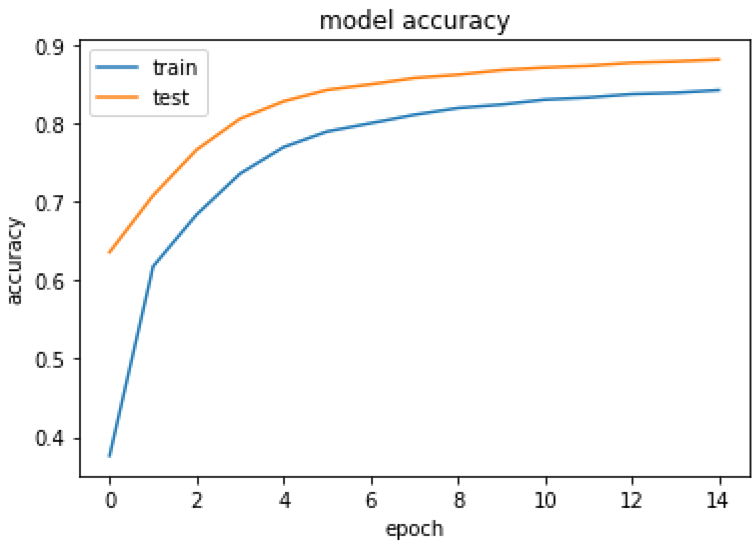
In [ ]:

```python
print(history.history.keys())
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

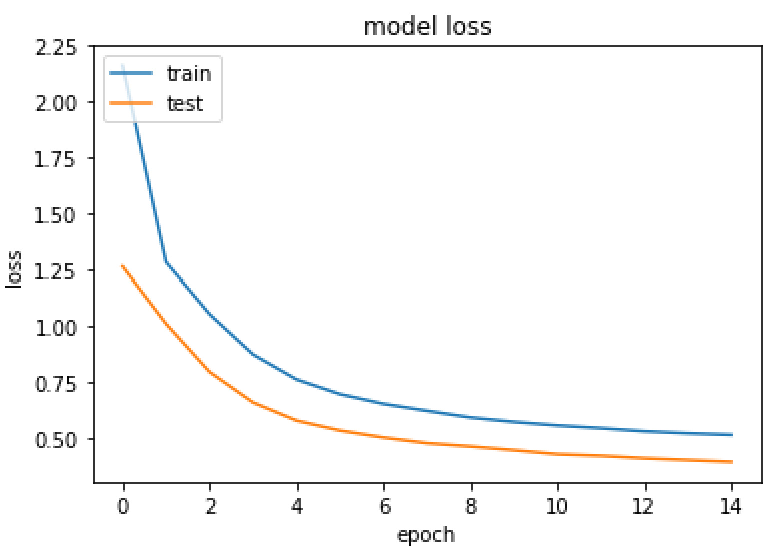
In [ ]:

```
# summarize history for accuracy

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



In [ ]:

```
# summarize history for loss

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



Experiment 4

Unfortunately, fit() does not return until training is complete. In order to avoid going down dead ends while adjusting your architecture and tuning its hyperparameters, you may prefer to visualize metrics during the process. TensorFlow includes the TensorBoard tool and the TensorBoard notebook extension for this purpose. Note: if you get a 403 error when trying to use TensorBoard in Google Colab, you may need to enable third-party cookies.

In [ ]:

```
# Load the TensorBoard notebook extension

%load_ext tensorboard
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

In [ ]:

```
import datetime
```

In [ ]:

```
# Clear any logs from previous runs
!rm -rf ./logs/
```

In [ ]:

```
def create_model():
  return tf.keras.models.Sequential([
    tf.keras.Input(shape=input_shape),
    tf.keras.layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
    tf.keras.layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(num_classes, activation="softmax")
  ])
```

In [ ]:

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

log_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1

model_.fit(x=x_train,
           y=y_train,
           epochs=5,
           validation_data=(x_validate, y_validate),
           callbacks=[tensorboard_callback])
```

```
Epoch 1/5
3250/3250 [==============================] - 72s 22ms/step - loss: 2.3362 - accuracy: 0.
3185 - val_loss: 0.9892 - val_accuracy: 0.7111
Epoch 2/5
3250/3250 [==============================] - 71s 22ms/step - loss: 1.0691 - accuracy: 0.
6771 - val_loss: 0.6733 - val_accuracy: 0.7998
Epoch 3/5
3250/3250 [==============================] - 71s 22ms/step - loss: 0.8180 - accuracy: 0.
7490 - val_loss: 0.5565 - val_accuracy: 0.8323
```

```
Epoch 4/5
3250/3250 [==============================] - 71s 22ms/step - loss: 0.7199 - accuracy: 0.
7790 - val_loss: 0.5131 - val_accuracy: 0.8458
Epoch 5/5
3250/3250 [==============================] - 74s 23ms/step - loss: 0.6789 - accuracy: 0.
7909 - val_loss: 0.4698 - val_accuracy: 0.8571
```

Out[ ]:
```
<tensorflow.python.keras.callbacks.History at 0x7fdd2f9e7f90>
```

In [ ]:
```
%tensorboard --logdir logs/fit
```

Experiment 5

Now that you have a baseline convolutional network for comparison, begin experimenting with alternative architectures, optimizers, and hyperparameters for the EMNIST Letters dataset.

How much can you improve the accuracy over Project 5? Project 5 - Experiment 5

Test Accuracy: 0.9183173179626465

Project 7 - Experiment 5

Test Accuracy: 0.9254326820373535

There was a small improvement in accuracy comparing project 5 and project 7.

In [ ]:
```
def create_model_():
  return tf.keras.models.Sequential([
    tf.keras.Input(shape=input_shape),
    tf.keras.layers.Conv2D(32, kernel_size=(3, 3), activation="relu", padding='same'),
    tf.keras.layers.Conv2D(32, kernel_size=(3, 3), activation="relu", padding='same'),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Conv2D(64, kernel_size=(3, 3), activation="relu", padding='same'),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(num_classes, activation="softmax")
  ])
```

In [ ]:
```
model_1 = create_model_()
model_1.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

log_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1

model_1.fit(x=x_train,
          y=y_train,
          epochs=25,
          validation_data=(x_validate, y_validate),
          callbacks=[tensorboard_callback])
```

```
Epoch 1/25
3250/3250 [==============================] - 232s 71ms/step - loss: 1.7022 - accuracy:
0.4963 - val_loss: 0.5001 - val_accuracy: 0.8466
Epoch 2/25
3250/3250 [==============================] - 230s 71ms/step - loss: 0.5330 - accuracy:
0.8327 - val_loss: 0.4204 - val_accuracy: 0.8688
Epoch 3/25
3250/3250 [==============================] - 228s 70ms/step - loss: 0.4681 - accuracy:
0.8520 - val_loss: 0.4030 - val_accuracy: 0.8737
Epoch 4/25
3250/3250 [==============================] - 228s 70ms/step - loss: 0.4320 - accuracy:
0.8624 - val_loss: 0.3702 - val_accuracy: 0.8850
Epoch 5/25
3250/3250 [==============================] - 228s 70ms/step - loss: 0.4006 - accuracy:
0.8723 - val_loss: 0.3410 - val_accuracy: 0.8932
Epoch 6/25
3250/3250 [==============================] - 228s 70ms/step - loss: 0.3818 - accuracy:
0.8790 - val_loss: 0.3181 - val_accuracy: 0.9017
Epoch 7/25
3250/3250 [==============================] - 228s 70ms/step - loss: 0.3490 - accuracy:
0.8872 - val_loss: 0.3160 - val_accuracy: 0.9017
Epoch 8/25
3250/3250 [==============================] - 228s 70ms/step - loss: 0.3442 - accuracy:
0.8905 - val_loss: 0.3039 - val_accuracy: 0.9056
Epoch 9/25
3250/3250 [==============================] - 225s 69ms/step - loss: 0.3251 - accuracy:
0.8954 - val_loss: 0.2861 - val_accuracy: 0.9116
Epoch 10/25
3250/3250 [==============================] - 226s 70ms/step - loss: 0.3124 - accuracy:
0.8981 - val_loss: 0.2881 - val_accuracy: 0.9098
Epoch 11/25
3250/3250 [==============================] - 226s 69ms/step - loss: 0.3037 - accuracy:
0.9014 - val_loss: 0.2740 - val_accuracy: 0.9162
Epoch 12/25
3250/3250 [==============================] - 225s 69ms/step - loss: 0.2978 - accuracy:
0.9020 - val_loss: 0.2728 - val_accuracy: 0.9154
Epoch 13/25
3250/3250 [==============================] - 225s 69ms/step - loss: 0.2888 - accuracy:
0.9059 - val_loss: 0.2671 - val_accuracy: 0.9173
Epoch 14/25
3250/3250 [==============================] - 224s 69ms/step - loss: 0.2729 - accuracy:
0.9110 - val_loss: 0.2594 - val_accuracy: 0.9188
Epoch 15/25
3250/3250 [==============================] - 221s 68ms/step - loss: 0.2737 - accuracy:
0.9105 - val_loss: 0.2534 - val_accuracy: 0.9207
Epoch 16/25
3250/3250 [==============================] - 222s 68ms/step - loss: 0.2630 - accuracy:
0.9135 - val_loss: 0.2483 - val_accuracy: 0.9226
Epoch 17/25
3250/3250 [==============================] - 222s 68ms/step - loss: 0.2585 - accuracy:
0.9150 - val_loss: 0.2412 - val_accuracy: 0.9263
Epoch 18/25
3250/3250 [==============================] - 221s 68ms/step - loss: 0.2504 - accuracy:
0.9173 - val_loss: 0.2418 - val_accuracy: 0.9249
Epoch 19/25
3250/3250 [==============================] - 222s 68ms/step - loss: 0.2444 - accuracy:
0.9195 - val_loss: 0.2396 - val_accuracy: 0.9253
Epoch 20/25
3250/3250 [==============================] - 223s 69ms/step - loss: 0.2383 - accuracy:
0.9199 - val_loss: 0.2392 - val_accuracy: 0.9259
Epoch 21/25
3250/3250 [==============================] - 222s 68ms/step - loss: 0.2361 - accuracy:
0.9219 - val_loss: 0.2377 - val_accuracy: 0.9256
Epoch 22/25
3250/3250 [==============================] - 221s 68ms/step - loss: 0.2318 - accuracy:
```

```
0.9222 - val_loss: 0.2331 - val_accuracy: 0.9273
Epoch 23/25
3250/3250 [==============================] - 220s 68ms/step - loss: 0.2310 - accuracy:
0.9216 - val_loss: 0.2307 - val_accuracy: 0.9291
Epoch 24/25
3250/3250 [==============================] - 220s 68ms/step - loss: 0.2279 - accuracy:
0.9223 - val_loss: 0.2287 - val_accuracy: 0.9300
Epoch 25/25
3250/3250 [==============================] - 220s 68ms/step - loss: 0.2227 - accuracy:
0.9252 - val_loss: 0.2284 - val_accuracy: 0.9285
```

Out[ ]:

```
<tensorflow.python.keras.callbacks.History at 0x7fdd2dde0690>
```

In [ ]:

```
%tensorboard --logdir logs/fit
```

```
Reusing TensorBoard on port 6006 (pid 8415), started 1:33:35 ago. (Use '!kill 8415' to k
ill it.)
```

Experiment 6

When finished, evaluate your results on the test set.

In [ ]:

```
score_ = model_1.evaluate(x_test, y_test, verbose=0)
print("Test loss: ", score_[0])
print("Test accuracy: ", score_[1])
```

```
Test loss:  0.23645161092281342
Test accuracy:  0.9254326820373535
```

Experiment 7

Use plt.imshow() to view some of the misclassified images and examine their labels.

Describe what you think might have gone wrong.

The CNN is still misclassifing images due to the variation of images (handwriting). Although I do not think variation is the only issue. It could be that I require longer training periods and more units along with additional hyperparameter tuning. Which take more time to run even using tesnorboard.

In [ ]:

```
predicted_test = np.array([])
predicted_test = model_1.predict(x_test[:])

fig1 = plt.figure(figsize=(30,30))

cols1 = 15
rows1 = 15

for i in range(1,rows1*cols1 + 1):
  if (np.argmax(predicted_test[i])+64) != (np.argmax(y_test[i])+64):
    fig1.add_subplot(rows1, cols1, i)
    plt.axis('off')
    plt.title("Misclassified: {}".format(chr(np.argmax(predicted_test[i])+64)))
    plt.imshow(x_test[i].reshape([28,28]), cmap='Greys_r', interpolation='none')
  else:
    fig1.add_subplot(rows1, cols1, i)
    plt.axis('off')
    plt.title("Class: {}".format(chr(np.argmax(y_test[i])+64)))
```

```
    plt.imshow(x_test[i].reshape([28,28]), cmap='Greys_r', interpolation='none')

plt.show()
```