Beta-project - Using AI to classify climbing problems

This project uses the Moonboard Database

TODO:

- use k-fold (https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RepeatedKFold.htm ?)
- make merged graphics
- re-test every model and significant variations in order to show the effect of different techniques and their combination
- Do a better under-sampling (just removing some samples from majority class, and test training with max 1000 images per class) (add some data augmentation?)

General data consideration

We start our project by importing relevant Python libraries, for data science, visualization and machine learning. We'll tackle explainability later.

In [1]:
```
import json
import os
import datetime

import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

import tensorflow as tf
import tensorflow.keras as keras
```

```
2024-09-03 09:50:44.237867: E external/local_xla/xla/stream_executor/cuda/cu
da_fft.cc:485] Unable to register cuFFT factory: Attempting to register fact
ory for plugin cuFFT when one has already been registered
2024-09-03 09:50:44.261853: E external/local_xla/xla/stream_executor/cuda/cu
da_dnn.cc:8454] Unable to register cuDNN factory: Attempting to register fac
tory for plugin cuDNN when one has already been registered
2024-09-03 09:50:44.267946: E external/local_xla/xla/stream_executor/cuda/cu
da_blas.cc:1452] Unable to register cuBLAS factory: Attempting to register f
actory for plugin cuBLAS when one has already been registered
2024-09-03 09:50:44.283394: I tensorflow/core/platform/cpu_feature_guard.cc:
210] This TensorFlow binary is optimized to use available CPU instructions i
n performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild
TensorFlow with the appropriate compiler flags.
2024-09-03 09:50:46.239578: W tensorflow/compiler/tf2tensorrt/utils/py_util
s.cc:38] TF-TRT Warning: Could not find TensorRT
```

Download and import data

We use an extracted database found on GitHub.

We first download everything and import raw data, except the "problems.json" that mostly contains duplicates of the Masters 2019 dataset.

We include the Mini MoonBoard dataset, as it contains many routes that are still relevant to our problem. The model we'll use will not take into account the size of the route but rather local patterns.

We only include columns that will be relevant for our classification problem, i.e. data that influence a route's grade.

In [ ]:
```
!wget https://github.com/spookykat/MoonBoard/files/13193317/problems_2023_01
!unzip problems_2023_01_30 -d problems_2023_01_30
```

In [2]:
```
columns = {
  "apiId": int,
  "name": str,
  "grade": 'category',
  "userGrade": 'category',
  "method": 'category',
  "holdsetup": 'category',
  "holdsets": 'object',
  "moves": 'object',
  "angle": int
}
column_names = list(columns.keys())

df = pd.DataFrame(columns=column_names)

for filename in os.listdir('problems_2023_01_30'):
  if filename == 'problems.json':
    continue
```

```
  with open(os.path.join('problems_2023_01_30', filename), 'r') as f:
    data = json.load(f)
    local_df = pd.DataFrame(data["data"])
    angle = int(filename.rstrip('.json').split()[-1])
    local_df['angle'] = angle if angle < 90 else 40
    df = pd.concat([df, local_df[column_names]])

df.drop_duplicates(keep='first', subset='apiId', inplace=True)
df.set_index('apiId', inplace=True)
```

Parse fields

Fields with foreign relations

In [3]:
```
df["holdsetup"] = df["holdsetup"].map(lambda x: x['apiId'])
df["holdsets"] = df["holdsets"].map(lambda sets: [el['apiId'] for el in sets
```

Moves: all holds of the route

There are three types of holds:

- Starter hold: where to put hands at the beginning
- Middle hold
- End hold: where to put both hands for at least 3 seconds at the end of the route

In [4]:
```
WIDTH = 11
HEIGHT = 18
NUM_HOLD_TYPES = 3

MOVES_SHAPE=(WIDTH, HEIGHT, NUM_HOLD_TYPES)

def parse_holds(moves):
  holds = np.zeros(MOVES_SHAPE, dtype=np.uint8)
  for hold in moves:
    description = hold['description']
    column = ord(description[0].upper()) - ord('A')
    row = int(description[1:]) - 1
    channel = 0
    if hold['isStart']:
      channel = 1
    if hold['isEnd']:
      channel = 2
    holds[column, row, channel] = 1
  return holds


df["moves"] = df["moves"].map(parse_holds)
```

Merging "grade" and "userGrade"

When a userGrade is present, it means that enough users rated this route so we assume this is a more objective metric than opener's grade attribution.

In [5]:
```
df["grade"] = df["userGrade"].combine_first(df["grade"]).astype('category')
df.drop("userGrade", axis=1, inplace=True)
```

Putting the right dtypes

In [6]:
```
for column in df.columns:
  df[column] = df[column].astype(columns[column])
```

Empty data

In [7]:
```
df.isna().sum()
```

Out[7]:
```
name         0
grade        0
method       0
holdsetup    0
holdsets     0
moves        0
angle        0
dtype: int64
```

There is no empty data, but one can uncomment the code below if any shows up.

In [8]:
```
# df.select_dtypes(include=[int, float]).fillna(df.mean(), inplace=True)
# categories = df.select_dtypes(include=['category', 'object'])
# categories.fillna(categories.mode().iloc[0])
```

Data Visualization and analysis

Overall information

In [9]:
```
df.describe()
```

Out[9]:

	angle
count	143100.000000
mean	38.548952
std	4.433996
min	25.000000
25%	40.000000
50%	40.000000
75%	40.000000
max	40.000000

In [10]:
```
df['holdsets'].value_counts()
```

Out[10]:
```
holdsets
[3, 4, 5]          32702
[4, 5]             24720
[4, 5, 8]          12873
[3, 4, 5, 8]        8615
[3, 4, 5, 8, 9]     4963
                   ...
[5, 9, 11]            53
[10]                  50
[5, 11]               47
[5, 9, 10]            43
[5, 10]               31
Name: count, Length: 78, dtype: int64
```

In [11]:
```
df['holdsetup'].value_counts()
```

Out[11]:
```
holdsetup
1     59506
15    55122
17    24802
19     3670
Name: count, dtype: int64
```

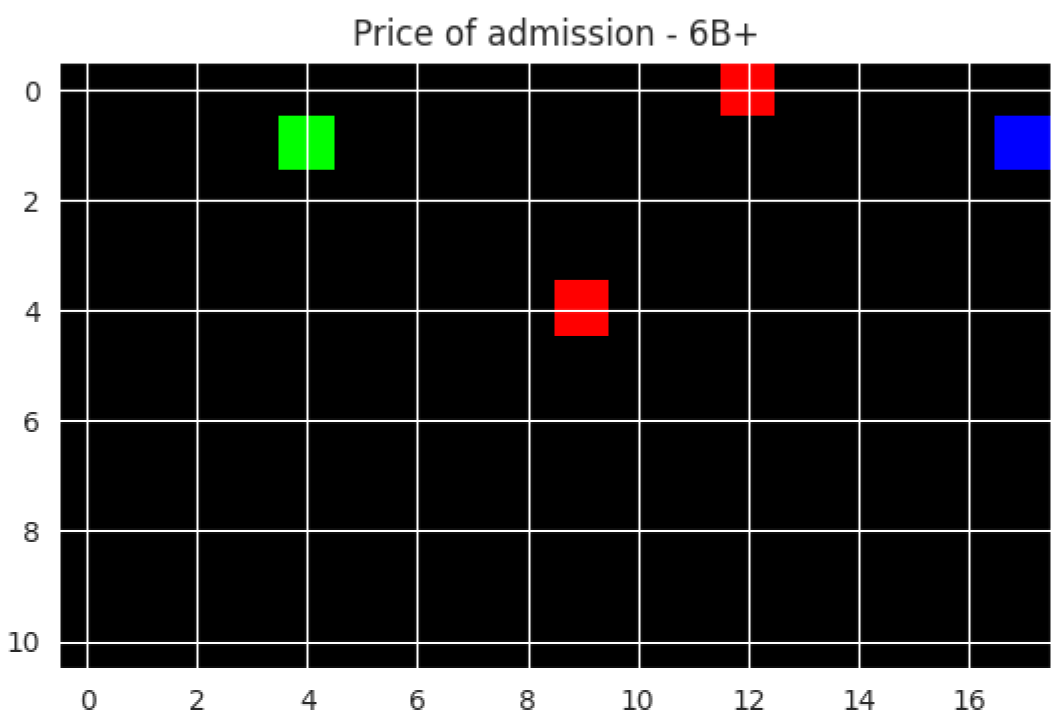
In [12]:
```
all_grades = list(df['grade'].cat.categories)
all_grades
```

Out[12]:
```
['5+',
 '6A',
 '6A+',
 '6B',
 '6B+',
 '6C',
 '6C+',
 '7A',
 '7A+',
 '7B',
 '7B+',
 '7C',
 '7C+',
 '8A',
 '8A+',
 '8B',
 '8B+']
```

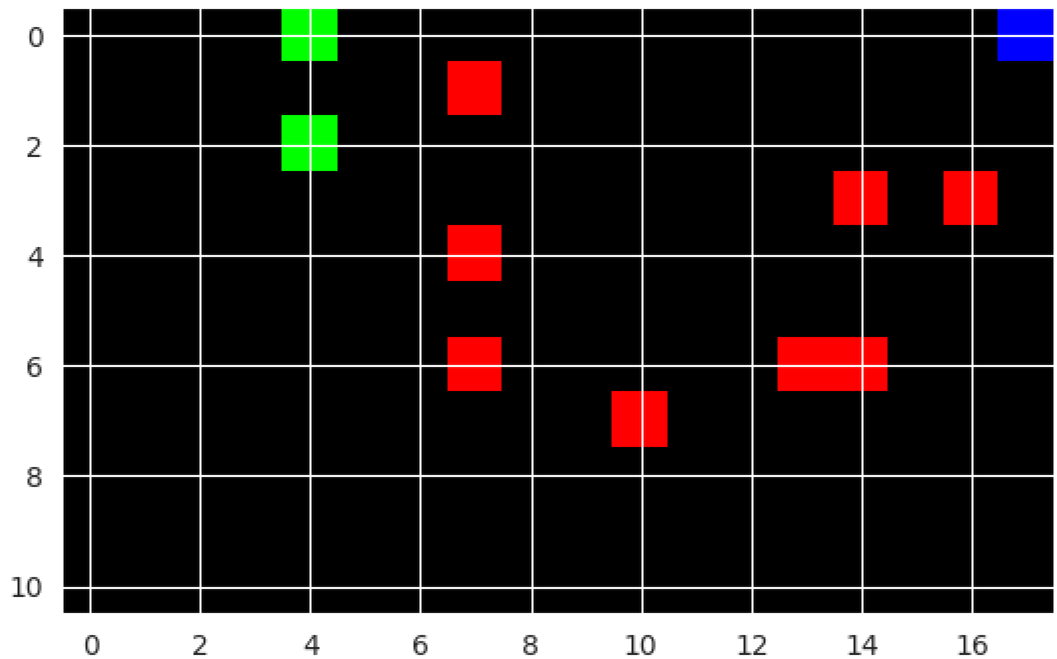
Visualize routes

We plot some of the routes to better visualize their structure. The plot will not be in the same orientation, due to the array structure: the left side is the bottom of the route.

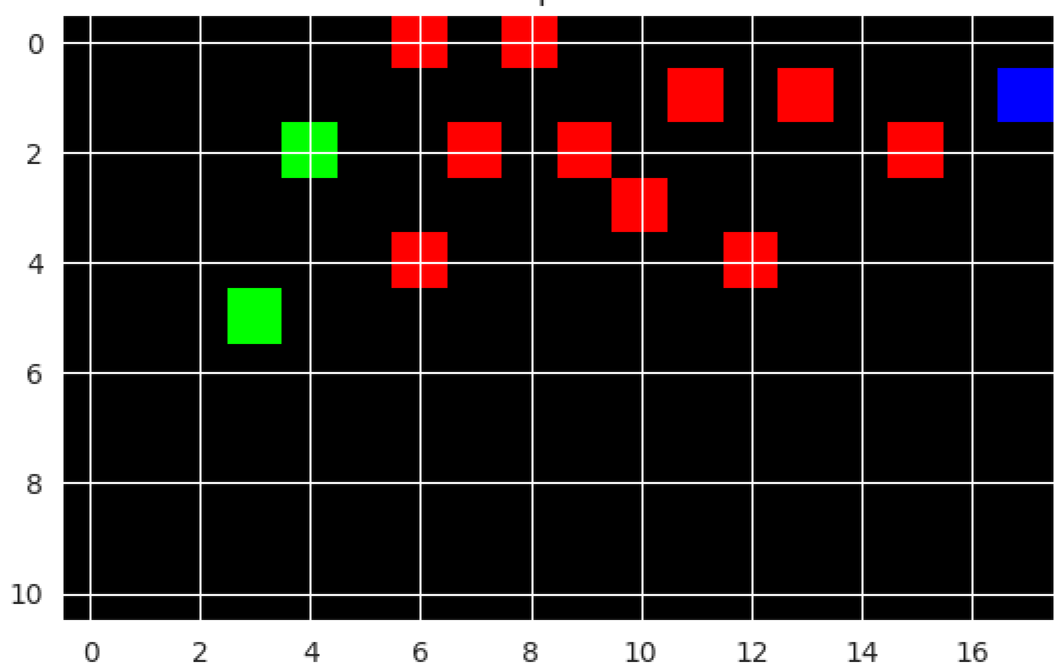
In [38]:
```
for _, route in df.sample(n=6).iterrows():
  plt.imshow(route['moves'] * 255)
  plt.title(f"{route['name']} - {route['grade']}")
  plt.show()
```



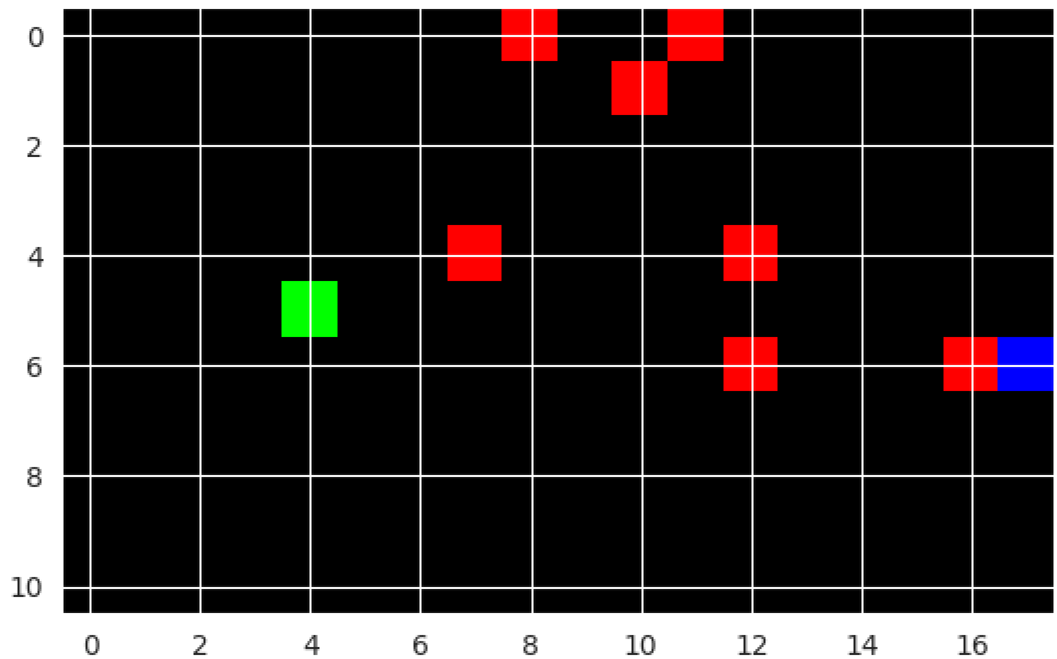
TRUMP KOOK - 6C+



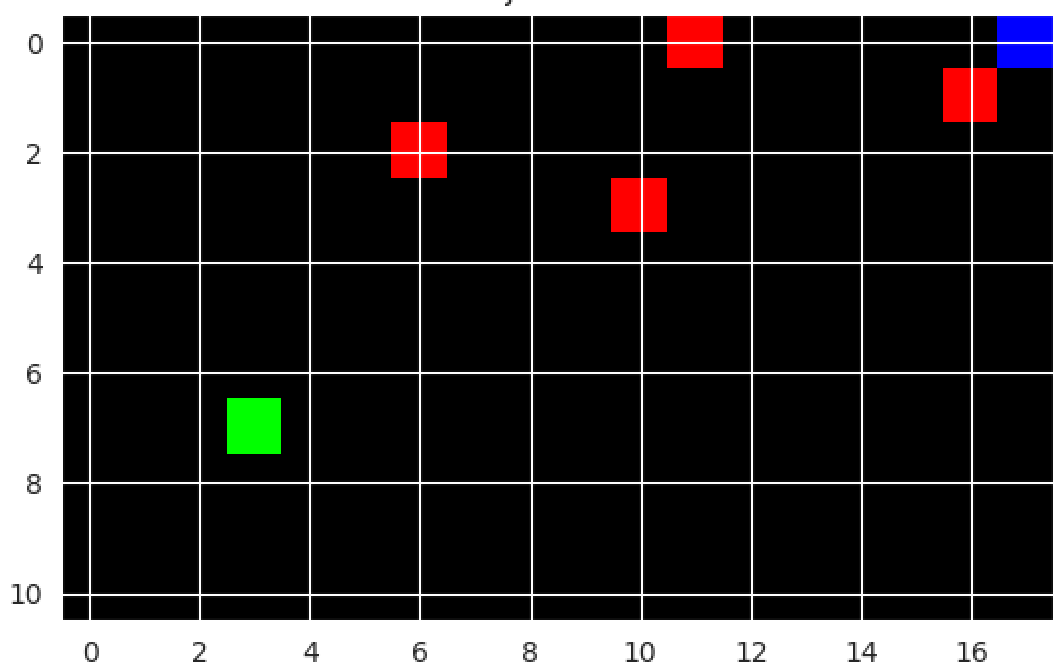
Static Gripz 15 - 6C+

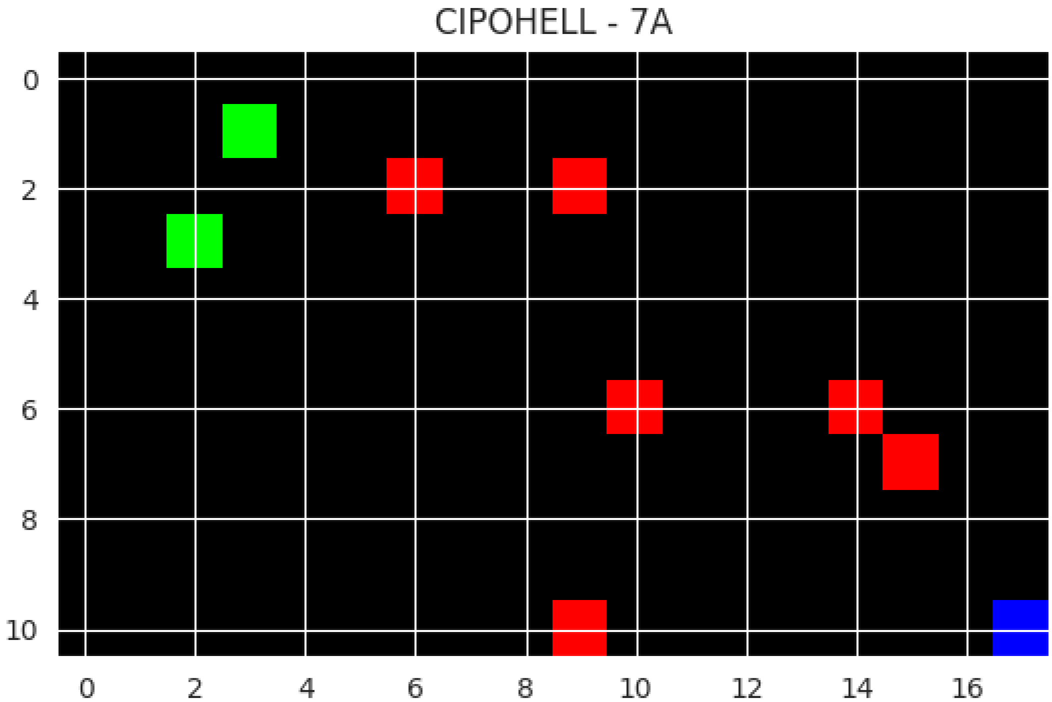


DOING IS BETTER - 6C+



Beaujolais - 7A+

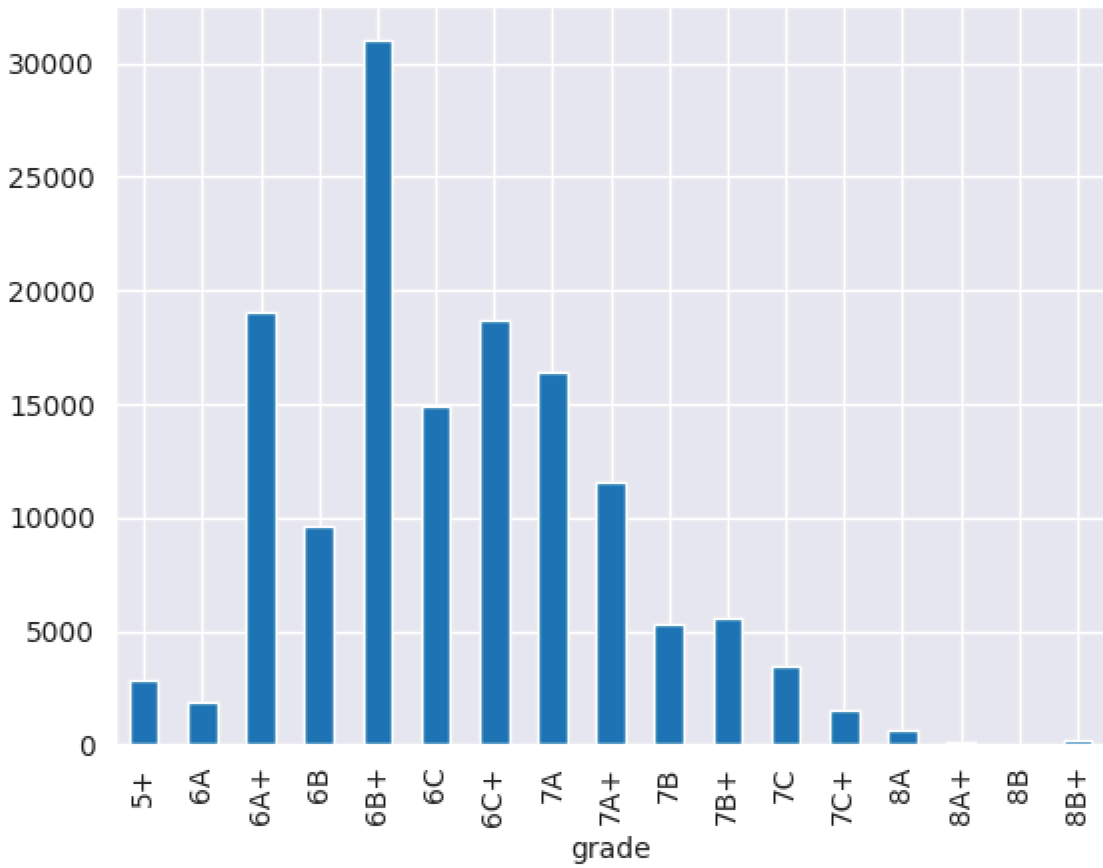




Plot the distribution of classes

```
In [59]: def plot_category_hist(dataframe, cat):
           dataframe[cat].value_counts().sort_index().plot(kind='bar')

         plot_category_hist(df, "grade")
```



Middle-grade routes are over-represented, with 6B+ routes being clearly omnipresent.

Check the influence of the resolution method

Let's see another field: "method", describing how the climber should physically solve the problem. This can significantly influence the difficulty and feasibility of a route.

In [41]:
```
df.groupby('method').size() / len(df)
```

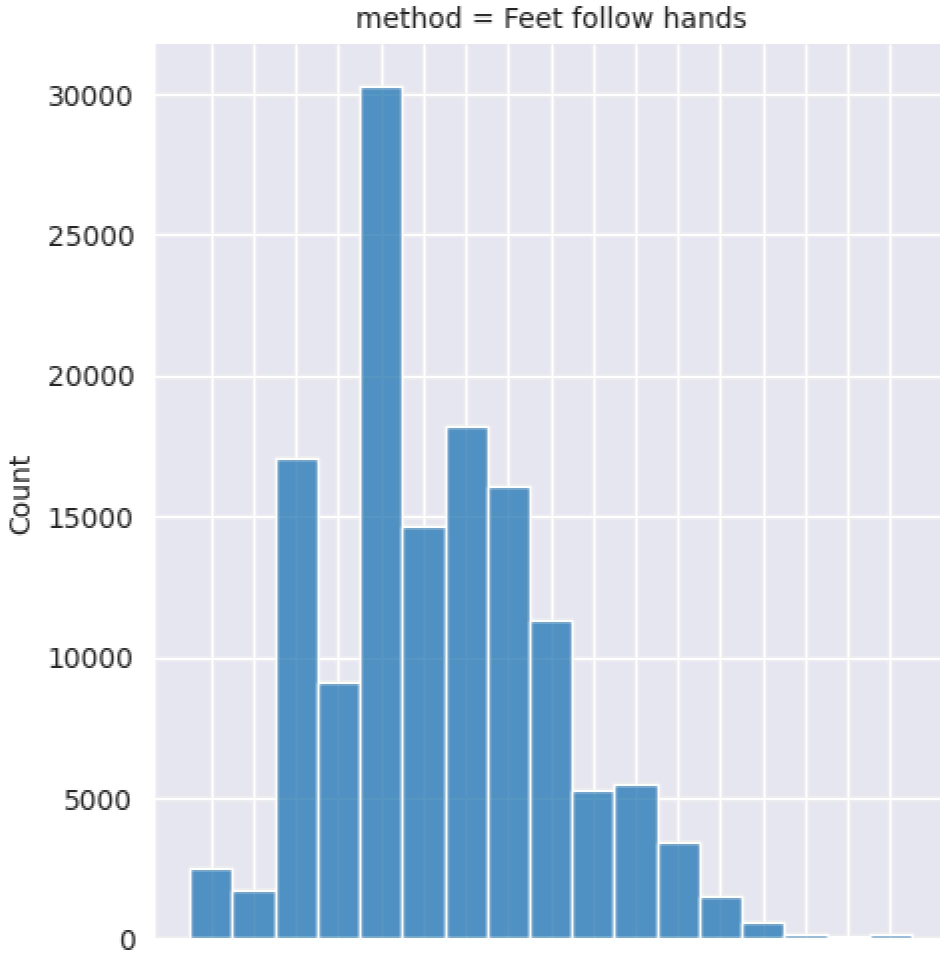
```
/tmp/ipykernel_6610/647761904.py:1: FutureWarning: The default of observed=F
alse is deprecated and will be changed to True in a future version of panda
s. Pass observed=False to retain current behavior or observed=True to adopt
the future default and silence this warning.
  df.groupby('method').size() / len(df)
```

Out[41]:
```
method
Feet follow hands                0.961579
Feet follow hands + screw ons    0.025618
Footless                         0.000021
Footless + kickboard             0.009553
Screw ons only                   0.003229
dtype: float64
```

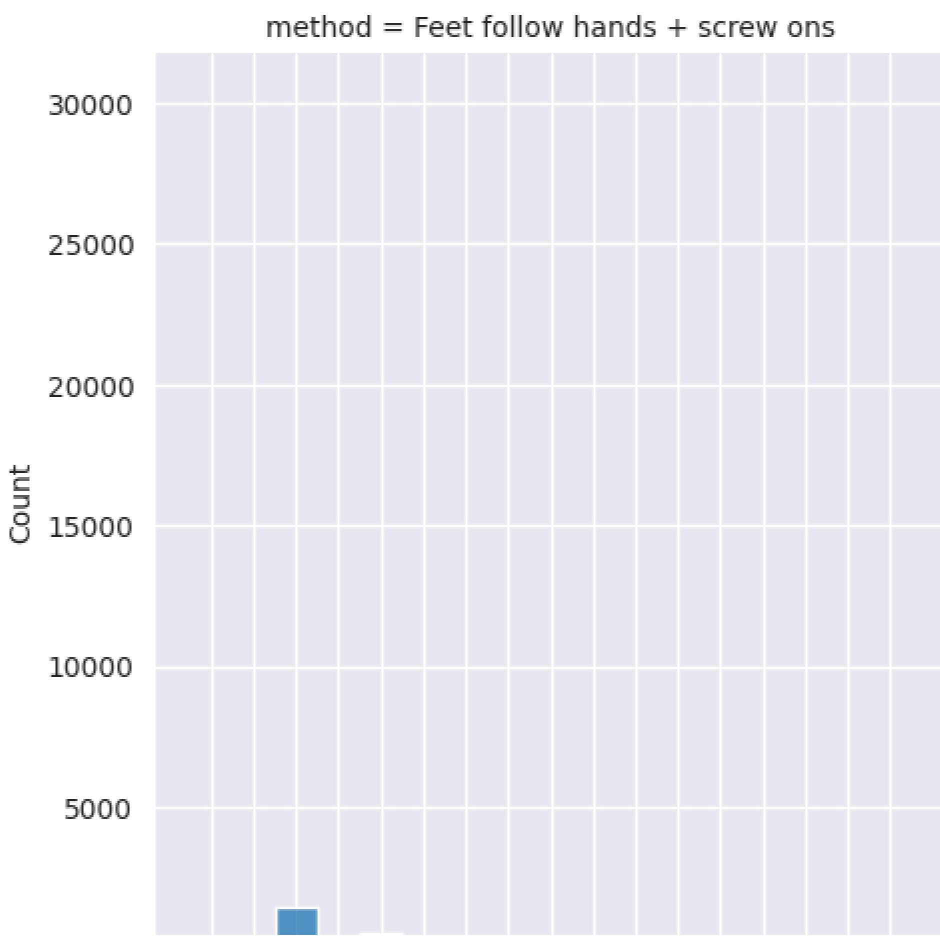
In [42]:
```python
nb_methods = len(df['method'].dtype.categories)

sns.displot(data=df, x='grade', row='method')
plt.xticks(rotation=80)
```

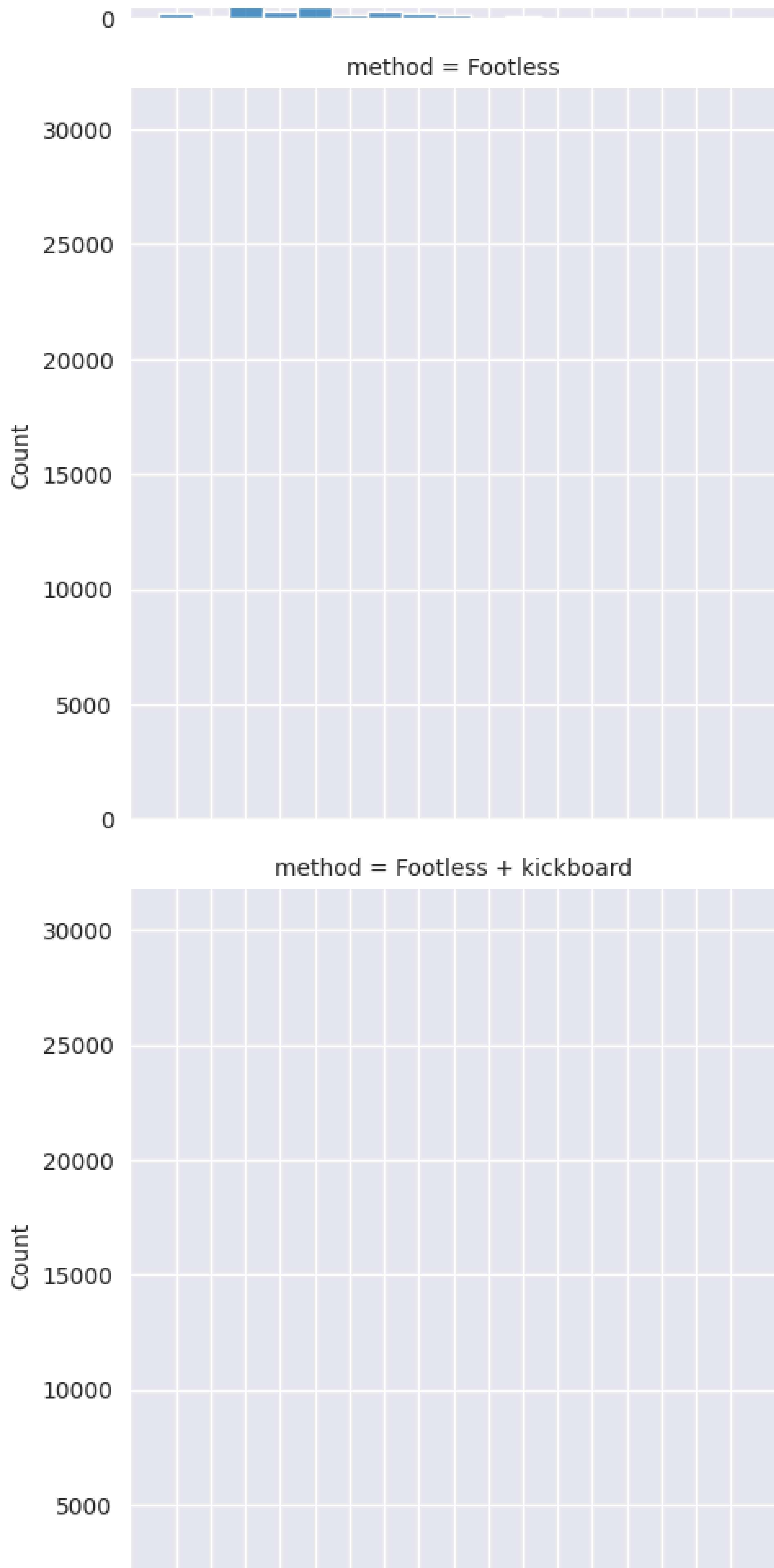
Out[42]:
```
([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16],
 [Text(0, 0, '5+'),
  Text(1, 0, '6A'),
  Text(2, 0, '6A+'),
  Text(3, 0, '6B'),
  Text(4, 0, '6B+'),
  Text(5, 0, '6C'),
  Text(6, 0, '6C+'),
  Text(7, 0, '7A'),
  Text(8, 0, '7A+'),
  Text(9, 0, '7B'),
  Text(10, 0, '7B+'),
  Text(11, 0, '7C'),
  Text(12, 0, '7C+'),
  Text(13, 0, '8A'),
  Text(14, 0, '8A+'),
  Text(15, 0, '8B'),
  Text(16, 0, '8B+')])
```



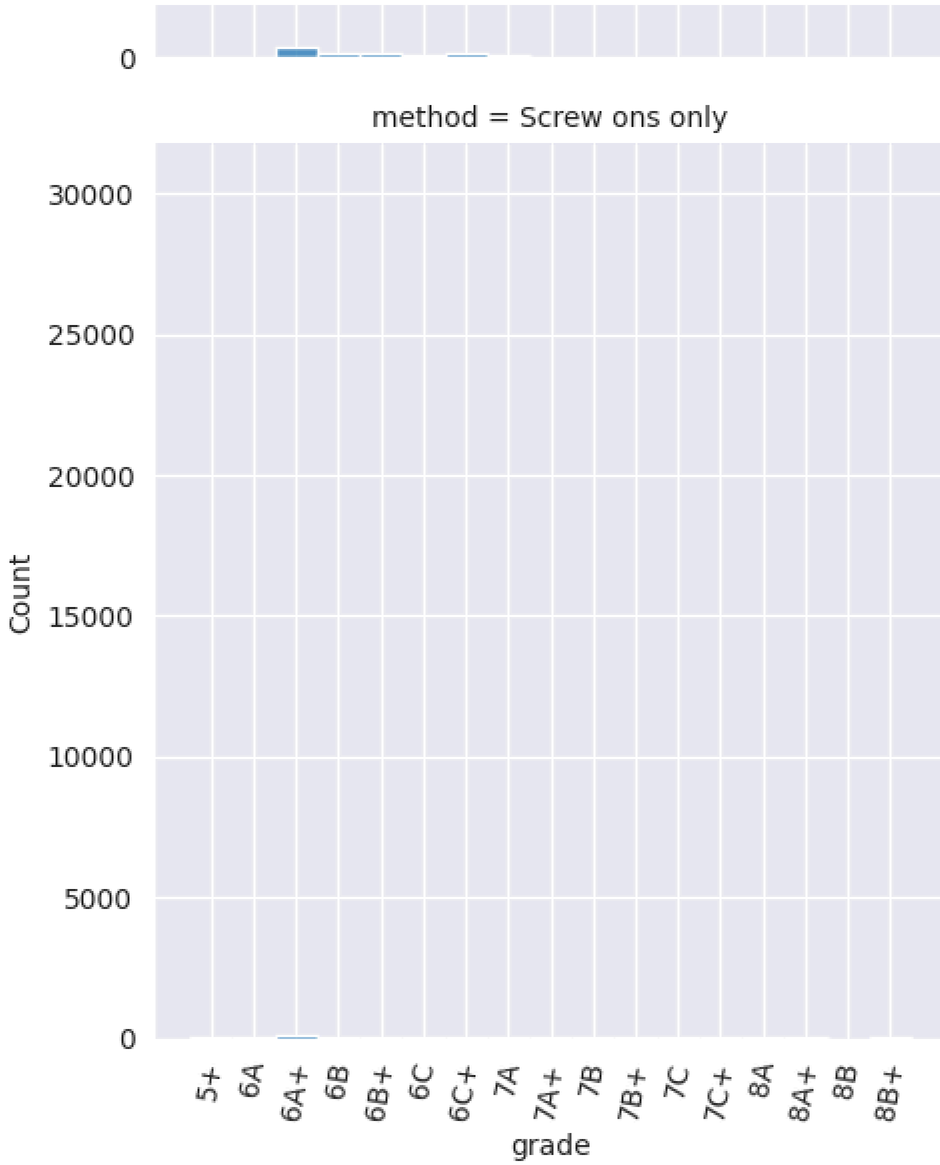
method = Feet follow hands
Count
30000
25000
20000
15000
10000
5000
0



method = Feet follow hands + screw ons
Count
30000
25000
20000
15000
10000
5000



0
method = Footless
30000
25000
20000
15000
10000
5000
0
Count
method = Footless + kickboard
30000
25000
20000
15000
10000
5000
Count



Data Preparation

We now want to prepare our data specifically for Machine Learning. We'll apply several well-known techniques, such as encoding (one-hot, rare...) and normalization, in order for our models to learn as much relevant data as possible to generalize well. We'll finish by splitting the data into training and testing sets - the validation set is created directly when training a new model.

Separate features and labels

We keep `moves` separate, as it's a special data type that we can tackle in many different ways. We'll see later two views: as a simple sequence, or as images.

In [13]:
```
features = df.drop(columns=['moves', 'grade', 'name'])
moves = df['moves']
labels = df['grade']
```

Rare encoding

For each categorical feature, we group together categories that appear less than 1% of the time.

For instance, the `method` feature has a clear imbalance: thus values `footless`, `footless + kickboard` and `screw ons only` must be gathered together. We can see this group as "difficult", as the climber is not allowed to use their feet for these routes.

In [14]:
```
rare_threshold = 0.1

for col in features.select_dtypes(include=['category']):
  value_counts = features[col].value_counts()
  rare_values = list(value_counts[value_counts / len(features) < rare_thresh
  if len(rare_values) > 1:
    features[col] = features[col].cat.add_categories(['Rare'])
    features.loc[features[col].isin(rare_values), col] = 'Rare'
    features[col] = features[col].cat.remove_unused_categories()
```

One-hot encoding

Before going further, we need to convert all features except 'moves' to numerical ones.

Dealing with the `holdsets` feature

This field contain arrays of holdsets, so we'll use a one-hot encoding to indicate for each holdset if a route contains it.

In [15]:
```
# TODO: use https://scikit-learn.org/stable/modules/preprocessing_targets.ht

def flatten_array(arr):
  result = []
  for el in arr:
    result.extend(el)
  return result

all_sets = list(features['holdsets'].value_counts().index)
all_sets = flatten_array(all_sets)
all_sets = list(set(all_sets))
```

```
for i in all_sets:
  features[f'holdset_{i}'] = False
```

In [16]:
```
def encode_sets(x):
  for i in x['holdsets']:
    x[f'holdset_{i}'] = True
  return x

features = features.apply(encode_sets, axis=1)
features.drop(columns=['holdsets'], inplace=True)
```

Dealing with other features

We use a basic one-hot encoding for other features, which is done quickly using pandas.

In [17]:
```
features['holdsetup'] = features['holdsetup'].astype('category')
features = pd.get_dummies(features)

labels = pd.get_dummies(labels)
```

Now that we have all the columns, we can get the number of distinct features and labels:

In [18]:
```
nb_labels = len(labels.columns)
nb_features = len(features.columns)
```

Normalization

We use min-max normalization, as the distribution of numerical inputs (i.e. `angle`) is not normal.

In [19]:
```
for col in features.select_dtypes(include=[int, float]):
  features[col] = (features[col] - features[col].min()) / (features[col].max
```

Split into train and test datasets

As explained earlier, the validation dataset will be created directly when fitting the model.

In [20]:
```
from sklearn.model_selection import train_test_split

test_split = 0.2

train_features, test_features, train_moves, test_moves, train_labels, test_l
  features,
  moves,
  labels,
```

```
  test_size=test_split,
  stratify=labels,
)

train_moves = np.stack(train_moves.values)
test_moves = np.stack(test_moves.values)
```

Helpers to build, train and analyse models

In this part, we define some helpers functions that we are going to reuse for all models : from building a model to analysing results.

Skeleton: build and fit model

We first define a function to plot a model, which can be useful to verify the connections made, especially as our models will not be sequential.

In [21]:
```
def plot_model(model):
  keras.utils.plot_model(
    model,
    show_shapes=True,
    show_dtype=True,
    show_layer_names=True,
    show_layer_activations=True,
    expand_nested=True
  )
```

For all models, we use the Categorical Crossentropy by default, as we face a multi-label classification problem, but where each object only has a unique class.

We also use different accuracy metrics not to rely only on the basic accuracy. Indeed, in the reality of climbing, it is okay to be one grade off - in many cases, the climber won't really notice the difference.

In [22]:
```
def compile_model(
        model=None,
        build_function=None,
        learning_rate=1e-3,
        loss=keras.losses.CategoricalCrossentropy(),
        metrics=None):

    if model is None:
        model = build_function()

    if metrics is None:
        metrics=[]
```

```
    metrics.extend([
      keras.metrics.CategoricalAccuracy(name='accuracy'),
      keras.metrics.TopKCategoricalAccuracy(k=3, name='accuracy_at_three'),
      keras.metrics.TopKCategoricalAccuracy(k=5, name='accuracy_at_five')
    ])

    model.compile(
        optimizer=keras.optimizers.RMSprop(learning_rate=learning_rate),
        loss=loss,
        metrics=metrics
    )
    return model
```

Finally, we create the function to train models. We use several callbacks to save our models at intermediary and "best-accuracy" stages, as well as exporting data for TensorBoard. This will allow us to precisely analyse our results and plot relevant graphs.

In [23]:

```
def train_model(model,
                training_features,
                training_labels,
                epochs=100,
                callbacks=None,
                early_stopping=None,
                validation_split=0.2,
                batch_size=64,
                class_weight=None,
                name='model',
                log_dir='logs/fit'
                ):

    if callbacks is None:
        callbacks = []

    if early_stopping is not None:
        callbacks.append(keras.callbacks.EarlyStopping(monitor='loss', patie

    date = datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
    callbacks.extend([
      keras.callbacks.ModelCheckpoint(f'models/{name}-{date}-{{epoch:02d}}-{
      keras.callbacks.ModelCheckpoint(f'models/{name}-{date}-best.keras', sa
      keras.callbacks.BackupAndRestore(backup_dir=f'/tmp/backup/{name}--{dat
      keras.callbacks.TensorBoard(log_dir=f'{log_dir}/{name}--{date}', histo
    ])

    model.fit(
      training_features,
      training_labels,
      epochs=epochs,
      callbacks=callbacks,
      validation_split=validation_split,
      batch_size=batch_size,
      class_weight=class_weight,
```

```
    )
    return model
```

Skeleton: overall accuracy

In [24]:
```
from sklearn.metrics import accuracy_score, balanced_accuracy_score
from sklearn.metrics import confusion_matrix
```

In [25]:
```
def parse_prediction(model, moves, features, labels):
  probabilities_labels = model.predict([moves, features], verbose=0)
  y_true = np.argmax(labels, axis=1)
  y_pred = np.argmax(probabilities_labels, axis=1)
  predicted = np.zeros_like(probabilities_labels).astype(bool)
  predicted[np.arange(probabilities_labels.shape[0]), y_pred] = True

  return predicted, y_true, y_pred


def load_best_model(
        name='model',
  ):
    best_model = keras.models.load_model(f'models/{name}-best.keras')
    train_predicted, train_y_true, train_y_pred = parse_prediction(best_mode
    test_predicted, test_y_true, test_y_pred = parse_prediction(best_model,

    return best_model, train_predicted, train_y_true, train_y_pred, test_pre
```

In [26]:
```
def print_accuracies(model, y_true, y_pred):
  metrics = model.evaluate([test_moves, test_features], test_labels, verbose
  print(f'Accuracy: {metrics[1] * 100:.2f}%')
  print(f'Balanced Accuracy: {balanced_accuracy_score(y_true, y_pred) * 100:
  print(f'Accuracy for Top3: {metrics[2] * 100:.2f}%')
  print(f'Accuracy for Top5: {metrics[3] * 100:.2f}%')
```

In [27]:
```
def confusion_matrix_analysis(y_true, y_pred):
  cm = confusion_matrix(y_true, y_pred)
  cm_normalized = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

  per_class_accuracy = cm.diagonal() / cm.sum(axis=1)
  for i, acc in enumerate(per_class_accuracy):
    print(f"Accuracy for grade {all_grades[i]}: {acc * 100:.2f}%")

  plt.figure(figsize=(15, 10))
  sns.heatmap(cm_normalized,  annot=True, fmt='.2f', cmap='rocket')
  plt.xlabel("Predicted Labels")
  plt.ylabel("True Labels")
  plt.title("Normalized Confusion Matrix")
  plt.show()
```

Skeleton: one-vs-rest analysis

In [28]:
```
def create_one_vs_rest_plot(xlabel, ylabel):
  fig, axs = plt.subplots(nrows=nb_labels, ncols=1, sharex=True)
  fig.set_size_inches(6, 4 * nb_labels)
  fig.text(0.5, 0.0005, xlabel, ha='center')
  fig.text(0.04, 0.5, ylabel, va='center', rotation='vertical')
  return fig, axs
```

In [29]:
```
from sklearn.metrics import roc_curve

def one_vs_rest_roc_curve(train_predicted, test_predicted):
  fig, axs = create_one_vs_rest_plot('False Positive Rate', 'True Positive R

  for i in range(nb_labels):
    train_fpr, train_tpr, _ = roc_curve(train_labels.values[:, i], train_pre
    test_fpr, test_tpr, _ = roc_curve(test_labels.values[:, i], test_predict
    plt.sca(axs[i])
    sns.lineplot(x=train_fpr, y=train_tpr, label='Train', ax=axs[i])
    sns.lineplot(x=test_fpr, y=test_tpr, label='Test', ax=axs[i])
    plt.ylabel(all_grades[i])

  fig.tight_layout()
```

In [30]:
```
from sklearn.metrics import precision_recall_curve

def one_vs_rest_precision_recall_curve(train_predicted, test_predicted):
  fig, axs = create_one_vs_rest_plot('False Positive Rate', 'True Positive R

  for i in range(nb_labels):
    train_precision, train_recall, _ = precision_recall_curve(train_labels.v
    test_precision, test_recall, _ = precision_recall_curve(test_labels.valu
    plt.sca(axs[i])
    sns.lineplot(x=train_precision, y=train_recall, ax=axs[i])
    sns.lineplot(x=test_precision, y=test_recall, ax=axs[i])
    plt.ylabel(all_grades[i])

  fig.tight_layout()
```

Baseline

For the baseline, we'll use a simple neural network with only Dense layers. We could even go simpler, but our final goal here is to implement a Deep Learning technique, so we only compare these kinds of algorithms.

In [75]:
```
def create_baseline():
  moves_inputs = keras.Input(shape=MOVES_SHAPE, name="moves")
  features_inputs = keras.Input(shape=(nb_features,), name="features")

  x = keras.layers.Flatten()(moves_inputs)

  x = keras.layers.concatenate([x, features_inputs])
```

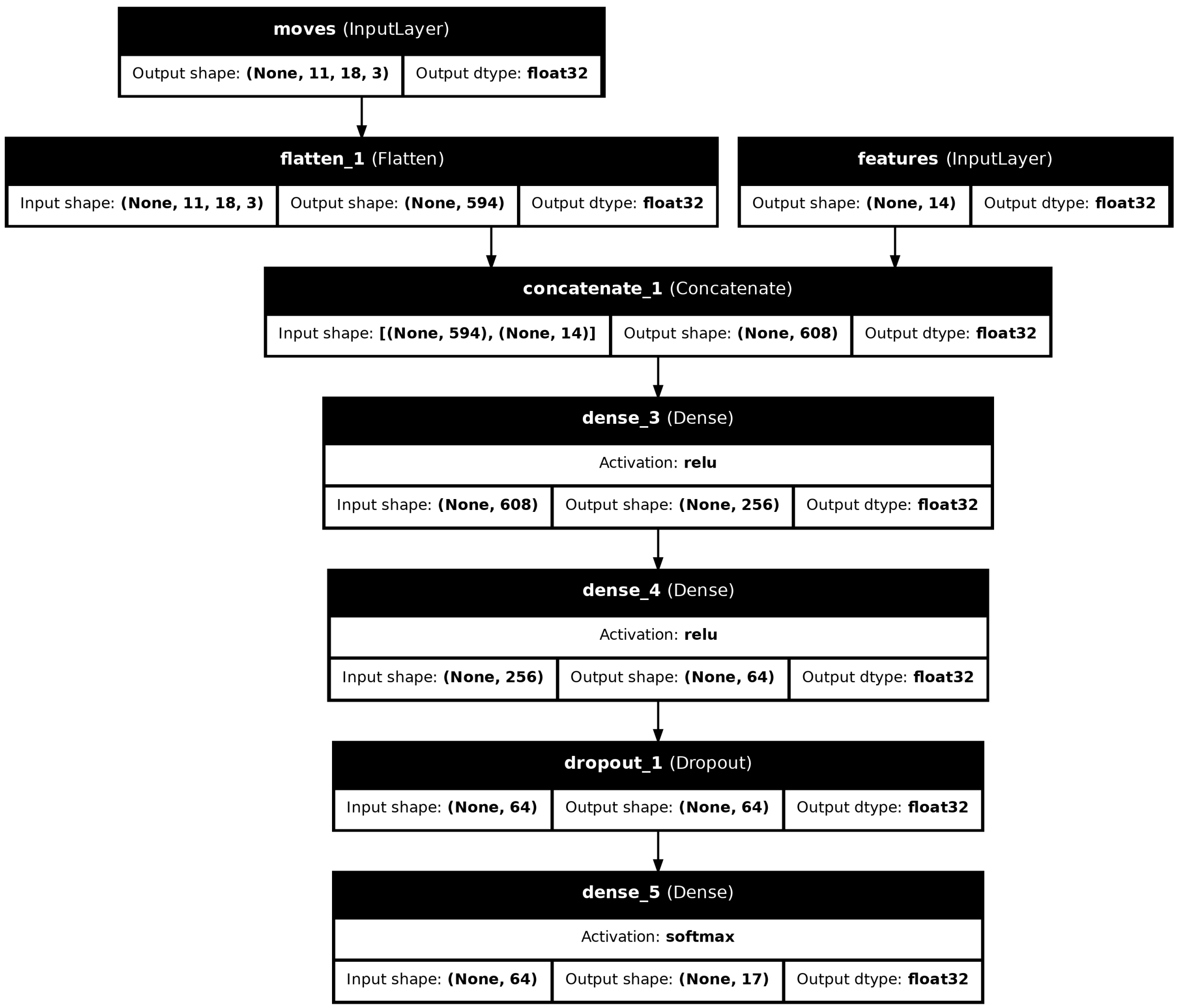
```python
  x = keras.layers.Dense(256, activation='relu')(x)
  x = keras.layers.Dense(64, activation='relu')(x)
  x = keras.layers.Dropout(0.5)(x)
  outputs = keras.layers.Dense(nb_labels, activation='softmax')(x)

  return keras.Model(inputs=[moves_inputs, features_inputs], outputs=outputs
```

In [ ]:
```python
baseline_model = compile_model(build_function=create_baseline)
```

In [29]:
```python
plot_model(baseline_model)
```

Out[29]:



Model training

In [34]:
```python
train_model(
  model=baseline_model,
  name='baseline',
  training_features=[train_moves, train_features],
  training_labels=train_labels,
  epochs=50,
  early_stopping=3
)
```

```
Epoch 1/50
```

```
WARNING: All log messages before absl::InitializeLog() is called are written
to STDERR
I0000 00:00:1724850969.850950   23742 service.cc:146] XLA service 0x7ff5b000
8670 initialized for platform CUDA (this does not guarantee that XLA will be
used). Devices:
I0000 00:00:1724850969.850978   23742 service.cc:154]   StreamExecutor devic
e (0): NVIDIA GeForce GTX 1050, Compute Capability 6.1
2024-08-28 15:16:09.909687: I tensorflow/compiler/mlir/tensorflow/utils/dump
_mlir_util.cc:268] disabling MLIR crash reproducer, set env var `MLIR_CRASH_
REPRODUCER_DIRECTORY` to enable.
2024-08-28 15:16:10.101404: I external/local_xla/xla/stream_executor/cuda/cu
da_dnn.cc:531] Loaded cuDNN version 8907
```

```
  10/1431 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.0963 - accuracy_at_
five: 0.4537 - accuracy_at_three: 0.3003 - loss: 2.7088
```

```
I0000 00:00:1724850971.684671   23742 device_compiler.h:188] Compiled cluste
r using XLA!  This line is logged at most once for the lifetime of the proce
ss.
```

```
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 11s 6ms/step - accuracy: 0.3135 - accuracy_at
_five: 0.8059 - accuracy_at_three: 0.6155 - loss: 1.9443 - val_accuracy: 0.1
963 - val_accuracy_at_five: 0.7147 - val_accuracy_at_three: 0.4818 - val_los
s: 2.3583
Epoch 2/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.3824 - accuracy_at_
five: 0.8994 - accuracy_at_three: 0.7355 - loss: 1.6275 - val_accuracy: 0.19
21 - val_accuracy_at_five: 0.6647 - val_accuracy_at_three: 0.4565 - val_los
s: 2.6150
Epoch 3/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.3977 - accuracy_at_
five: 0.9130 - accuracy_at_three: 0.7521 - loss: 1.5895 - val_accuracy: 0.19
26 - val_accuracy_at_five: 0.6792 - val_accuracy_at_three: 0.4675 - val_los
s: 2.5794
Epoch 4/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - accuracy: 0.4041 - accuracy_at
_five: 0.9190 - accuracy_at_three: 0.7638 - loss: 1.5662 - val_accuracy: 0.1
891 - val_accuracy_at_five: 0.6795 - val_accuracy_at_three: 0.4618 - val_los
s: 2.6829
Epoch 5/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.4078 - accuracy_at_
five: 0.9254 - accuracy_at_three: 0.7759 - loss: 1.5473 - val_accuracy: 0.18
89 - val_accuracy_at_five: 0.6700 - val_accuracy_at_three: 0.4579 - val_los
s: 2.7862
Epoch 6/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.4149 - accuracy_at_
five: 0.9299 - accuracy_at_three: 0.7811 - loss: 1.5275 - val_accuracy: 0.18
58 - val_accuracy_at_five: 0.6563 - val_accuracy_at_three: 0.4494 - val_los
s: 2.9607
Epoch 7/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.4232 - accuracy_at_
five: 0.9321 - accuracy_at_three: 0.7879 - loss: 1.5125 - val_accuracy: 0.18
92 - val_accuracy_at_five: 0.6665 - val_accuracy_at_three: 0.4565 - val_los
s: 2.9679
Epoch 8/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.4289 - accuracy_at_
five: 0.9358 - accuracy_at_three: 0.7914 - loss: 1.4951 - val_accuracy: 0.18
59 - val_accuracy_at_five: 0.6587 - val_accuracy_at_three: 0.4507 - val_los
s: 3.2283
Epoch 9/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.4316 - accuracy_at_
five: 0.9408 - accuracy_at_three: 0.8007 - loss: 1.4754 - val_accuracy: 0.18
66 - val_accuracy_at_five: 0.6665 - val_accuracy_at_three: 0.4547 - val_los
s: 3.1703
Epoch 10/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.4334 - accuracy_at_
five: 0.9407 - accuracy_at_three: 0.8041 - loss: 1.4743 - val_accuracy: 0.17
68 - val_accuracy_at_five: 0.6310 - val_accuracy_at_three: 0.4320 - val_los
s: 3.7468
Epoch 11/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.4342 - accuracy_at_
five: 0.9428 - accuracy_at_three: 0.8077 - loss: 1.4582 - val_accuracy: 0.18
75 - val_accuracy_at_five: 0.6702 - val_accuracy_at_three: 0.4572 - val_los
s: 3.5942
Epoch 12/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.4424 - accuracy_at_
```

```
five: 0.9438 - accuracy_at_three: 0.8131 - loss: 1.4521 - val_accuracy: 0.18
09 - val_accuracy_at_five: 0.6436 - val_accuracy_at_three: 0.4350 - val_los
s: 3.8996
Epoch 13/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.4435 - accuracy_at_
five: 0.9461 - accuracy_at_three: 0.8139 - loss: 1.4516 - val_accuracy: 0.17
76 - val_accuracy_at_five: 0.6409 - val_accuracy_at_three: 0.4348 - val_los
s: 3.8287
Epoch 14/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.4512 - accuracy_at_
five: 0.9491 - accuracy_at_three: 0.8213 - loss: 1.4337 - val_accuracy: 0.18
36 - val_accuracy_at_five: 0.6557 - val_accuracy_at_three: 0.4433 - val_los
s: 4.0063
Epoch 15/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.4497 - accuracy_at_
five: 0.9480 - accuracy_at_three: 0.8188 - loss: 1.4314 - val_accuracy: 0.18
18 - val_accuracy_at_five: 0.6515 - val_accuracy_at_three: 0.4441 - val_los
s: 4.0302
Epoch 16/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.4504 - accuracy_at_
five: 0.9500 - accuracy_at_three: 0.8239 - loss: 1.4224 - val_accuracy: 0.17
58 - val_accuracy_at_five: 0.6342 - val_accuracy_at_three: 0.4276 - val_los
s: 4.6645
Epoch 17/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.4540 - accuracy_at_
five: 0.9517 - accuracy_at_three: 0.8273 - loss: 1.4196 - val_accuracy: 0.17
72 - val_accuracy_at_five: 0.6418 - val_accuracy_at_three: 0.4306 - val_los
s: 4.6235
Epoch 18/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.4575 - accuracy_at_
five: 0.9517 - accuracy_at_three: 0.8272 - loss: 1.4199 - val_accuracy: 0.17
02 - val_accuracy_at_five: 0.6098 - val_accuracy_at_three: 0.4122 - val_los
s: 4.7977
Epoch 19/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.4570 - accuracy_at_
five: 0.9542 - accuracy_at_three: 0.8294 - loss: 1.4105 - val_accuracy: 0.18
03 - val_accuracy_at_five: 0.6486 - val_accuracy_at_three: 0.4388 - val_los
s: 4.7834
Epoch 20/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.4566 - accuracy_at_
five: 0.9519 - accuracy_at_three: 0.8274 - loss: 1.4132 - val_accuracy: 0.17
30 - val_accuracy_at_five: 0.6279 - val_accuracy_at_three: 0.4222 - val_los
s: 5.1380
Epoch 21/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.4598 - accuracy_at_
five: 0.9530 - accuracy_at_three: 0.8322 - loss: 1.4076 - val_accuracy: 0.17
70 - val_accuracy_at_five: 0.6233 - val_accuracy_at_three: 0.4191 - val_los
s: 5.1202
Epoch 22/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.4629 - accuracy_at_
five: 0.9539 - accuracy_at_three: 0.8322 - loss: 1.4022 - val_accuracy: 0.16
96 - val_accuracy_at_five: 0.6181 - val_accuracy_at_three: 0.4147 - val_los
s: 5.5360
Epoch 23/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.4628 - accuracy_at_
five: 0.9533 - accuracy_at_three: 0.8319 - loss: 1.3999 - val_accuracy: 0.17
```

```
60 - val_accuracy_at_five: 0.6259 - val_accuracy_at_three: 0.4242 - val_los
s: 6.0724
Epoch 24/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.4626 - accuracy_at_
five: 0.9547 - accuracy_at_three: 0.8318 - loss: 1.4069 - val_accuracy: 0.17
70 - val_accuracy_at_five: 0.6297 - val_accuracy_at_three: 0.4298 - val_los
s: 6.2287
Epoch 25/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.4664 - accuracy_at_
five: 0.9551 - accuracy_at_three: 0.8321 - loss: 1.4039 - val_accuracy: 0.16
94 - val_accuracy_at_five: 0.5998 - val_accuracy_at_three: 0.4060 - val_los
s: 6.4303
Epoch 26/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.4634 - accuracy_at_
five: 0.9544 - accuracy_at_three: 0.8337 - loss: 1.3987 - val_accuracy: 0.16
33 - val_accuracy_at_five: 0.5846 - val_accuracy_at_three: 0.3973 - val_los
s: 7.3381
Epoch 27/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.4626 - accuracy_at_
five: 0.9549 - accuracy_at_three: 0.8333 - loss: 1.3898 - val_accuracy: 0.16
76 - val_accuracy_at_five: 0.5965 - val_accuracy_at_three: 0.4040 - val_los
s: 7.0192
Epoch 28/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.4681 - accuracy_at_
five: 0.9562 - accuracy_at_three: 0.8350 - loss: 1.3892 - val_accuracy: 0.17
67 - val_accuracy_at_five: 0.6148 - val_accuracy_at_three: 0.4196 - val_los
s: 7.1313
Epoch 29/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.4664 - accuracy_at_
five: 0.9551 - accuracy_at_three: 0.8342 - loss: 1.3950 - val_accuracy: 0.17
02 - val_accuracy_at_five: 0.5947 - val_accuracy_at_three: 0.4061 - val_los
s: 7.5910
Epoch 30/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.4715 - accuracy_at_
five: 0.9562 - accuracy_at_three: 0.8356 - loss: 1.3937 - val_accuracy: 0.17
76 - val_accuracy_at_five: 0.6086 - val_accuracy_at_three: 0.4165 - val_los
s: 7.8518
Epoch 31/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.4664 - accuracy_at_
five: 0.9548 - accuracy_at_three: 0.8356 - loss: 1.3753 - val_accuracy: 0.17
02 - val_accuracy_at_five: 0.5980 - val_accuracy_at_three: 0.4044 - val_los
s: 8.3708
Epoch 32/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 11s 5ms/step - accuracy: 0.4660 - accuracy_at
_five: 0.9552 - accuracy_at_three: 0.8339 - loss: 1.3894 - val_accuracy: 0.1
775 - val_accuracy_at_five: 0.6171 - val_accuracy_at_three: 0.4239 - val_los
s: 7.9574
Epoch 33/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.4694 - accuracy_at_
five: 0.9561 - accuracy_at_three: 0.8365 - loss: 1.3852 - val_accuracy: 0.16
30 - val_accuracy_at_five: 0.5817 - val_accuracy_at_three: 0.3919 - val_los
s: 8.9698
Epoch 34/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - accuracy: 0.4682 - accuracy_at
_five: 0.9558 - accuracy_at_three: 0.8373 - loss: 1.3811 - val_accuracy: 0.1
713 - val_accuracy_at_five: 0.5955 - val_accuracy_at_three: 0.4041 - val_los
```

```
s: 9.4815
Epoch 35/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 11s 6ms/step - accuracy: 0.4713 - accuracy_at_five: 0.9556 - accuracy_at_three: 0.8345 - loss: 1.3875 - val_accuracy: 0.1697 - val_accuracy_at_five: 0.5869 - val_accuracy_at_three: 0.3981 - val_loss: 9.4306
Epoch 36/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.4695 - accuracy_at_five: 0.9550 - accuracy_at_three: 0.8384 - loss: 1.3788 - val_accuracy: 0.1722 - val_accuracy_at_five: 0.6066 - val_accuracy_at_three: 0.4096 - val_loss: 9.3496
Epoch 37/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.4673 - accuracy_at_five: 0.9545 - accuracy_at_three: 0.8339 - loss: 1.3723 - val_accuracy: 0.1766 - val_accuracy_at_five: 0.6145 - val_accuracy_at_three: 0.4185 - val_loss: 9.6869
Epoch 38/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.4746 - accuracy_at_five: 0.9559 - accuracy_at_three: 0.8409 - loss: 1.3745 - val_accuracy: 0.1750 - val_accuracy_at_five: 0.6019 - val_accuracy_at_three: 0.4125 - val_loss: 10.2297
Epoch 39/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.4723 - accuracy_at_five: 0.9560 - accuracy_at_three: 0.8385 - loss: 1.3769 - val_accuracy: 0.1672 - val_accuracy_at_five: 0.5978 - val_accuracy_at_three: 0.4046 - val_loss: 10.4519
Epoch 40/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.4705 - accuracy_at_five: 0.9552 - accuracy_at_three: 0.8385 - loss: 1.3687 - val_accuracy: 0.1719 - val_accuracy_at_five: 0.6022 - val_accuracy_at_three: 0.4091 - val_loss: 10.6766
Epoch 41/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.4742 - accuracy_at_five: 0.9543 - accuracy_at_three: 0.8379 - loss: 1.3718 - val_accuracy: 0.1688 - val_accuracy_at_five: 0.5942 - val_accuracy_at_three: 0.4047 - val_loss: 11.5640
```

Out[34]: <Functional name=functional_1, built=True>

In [42]:
```
baseline_model, train_predicted, train_y_true, train_y_pred, test_predicted,
```

Accuracy

In [36]:
```
print_accuracies(baseline_model, test_y_true, test_y_pred)
```

```
Accuracy: 33.98%
Balanced Accuracy: 17.41%
Accuracy for Top3: 68.53%
Accuracy for Top5: 86.47%
```

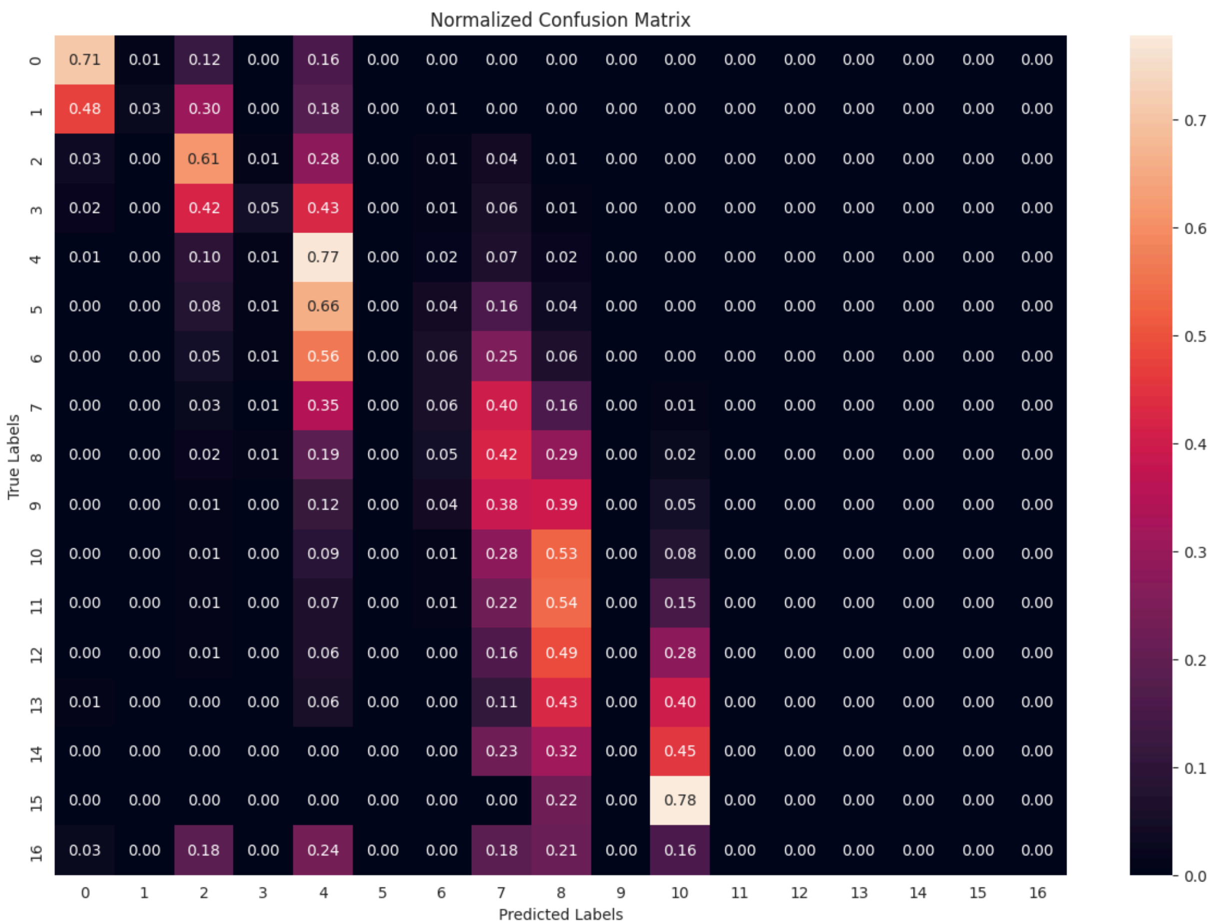
Our model is overfitting straight away: it's not able to detect any patterns, so it learns the training dataset by heart.

Confusion matrix

In [112…

```
confusion_matrix_analysis(test_y_true, test_y_pred)
```

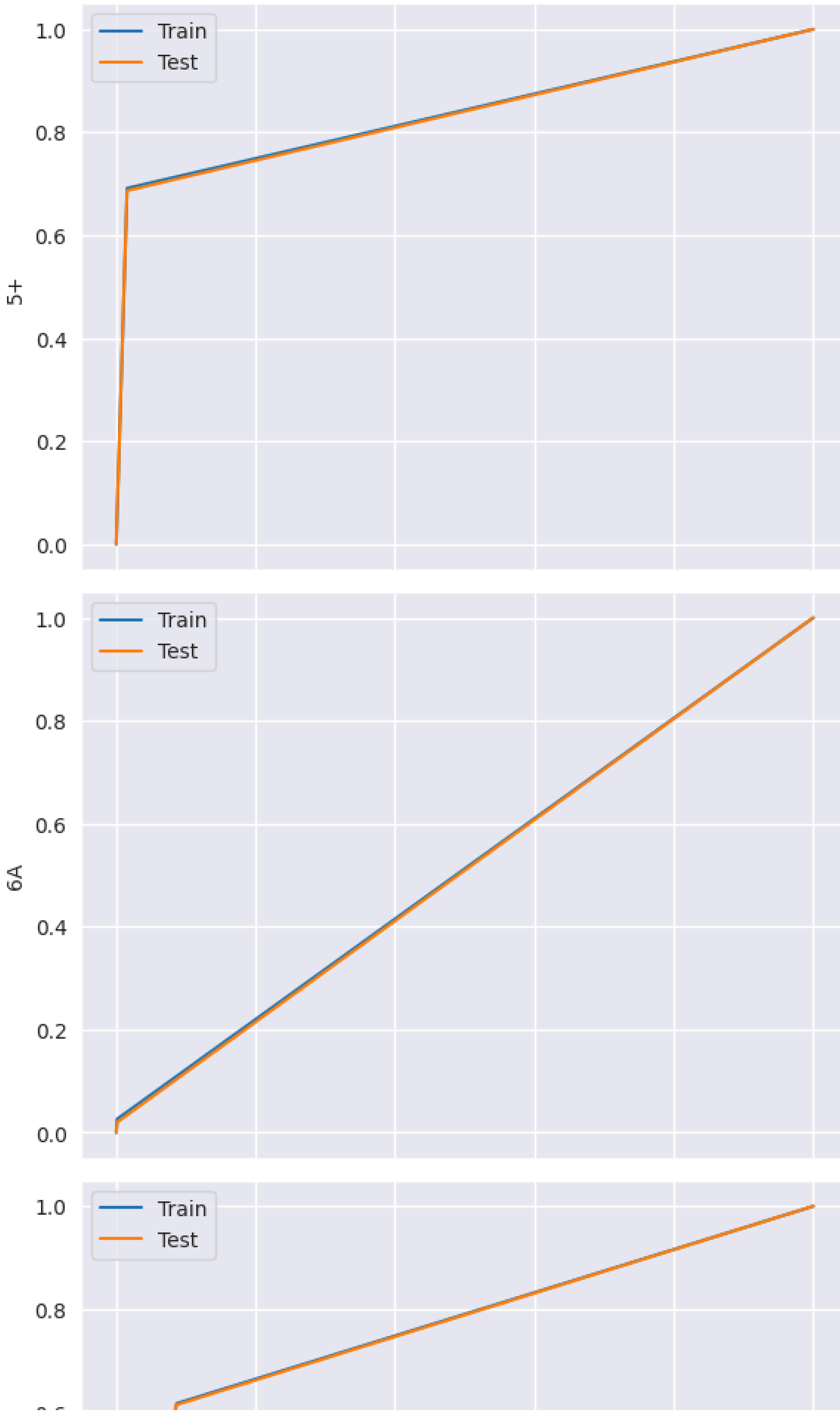
```
Accuracy for grade 5+: 70.52%
Accuracy for grade 6A: 2.99%
Accuracy for grade 6A+: 61.38%
Accuracy for grade 6B: 4.88%
Accuracy for grade 6B+: 77.37%
Accuracy for grade 6C: 0.00%
Accuracy for grade 6C+: 5.68%
Accuracy for grade 7A: 39.93%
Accuracy for grade 7A+: 29.04%
Accuracy for grade 7B: 0.00%
Accuracy for grade 7B+: 7.77%
Accuracy for grade 7C: 0.00%
Accuracy for grade 7C+: 0.00%
Accuracy for grade 8A: 0.00%
Accuracy for grade 8A+: 0.00%
Accuracy for grade 8B: 0.00%
Accuracy for grade 8B+: 0.00%
```

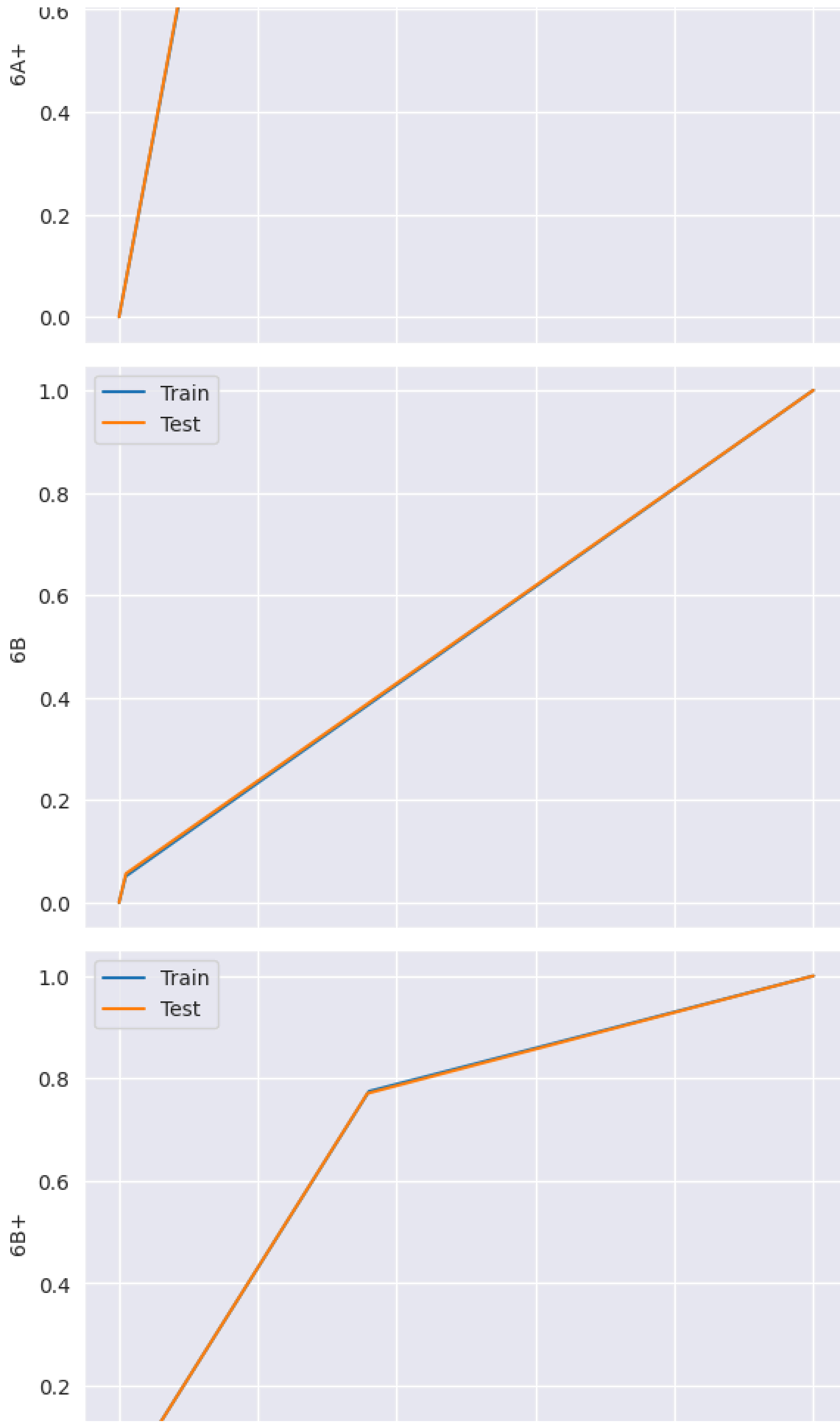


One-vs-rest analysis

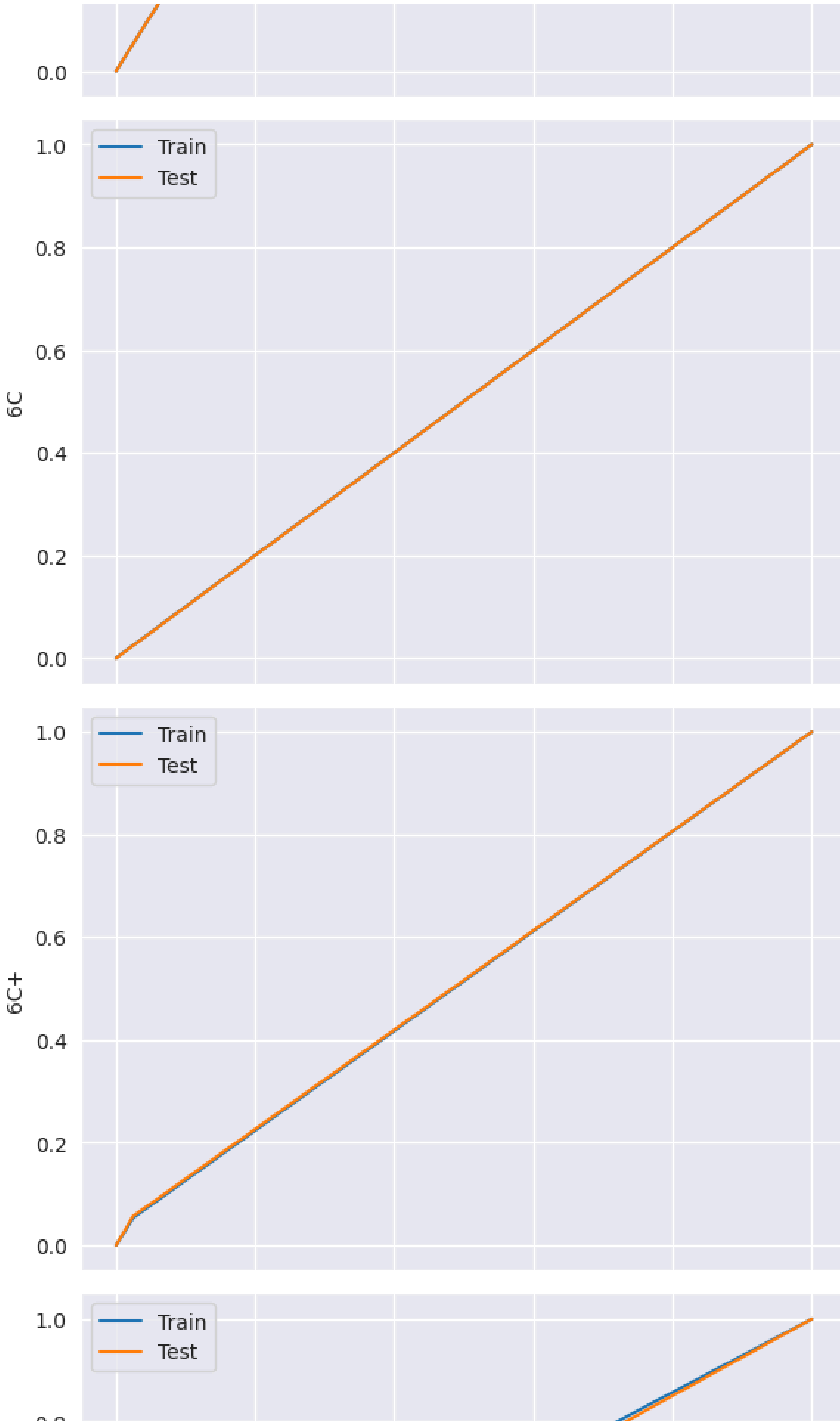
In [51]:

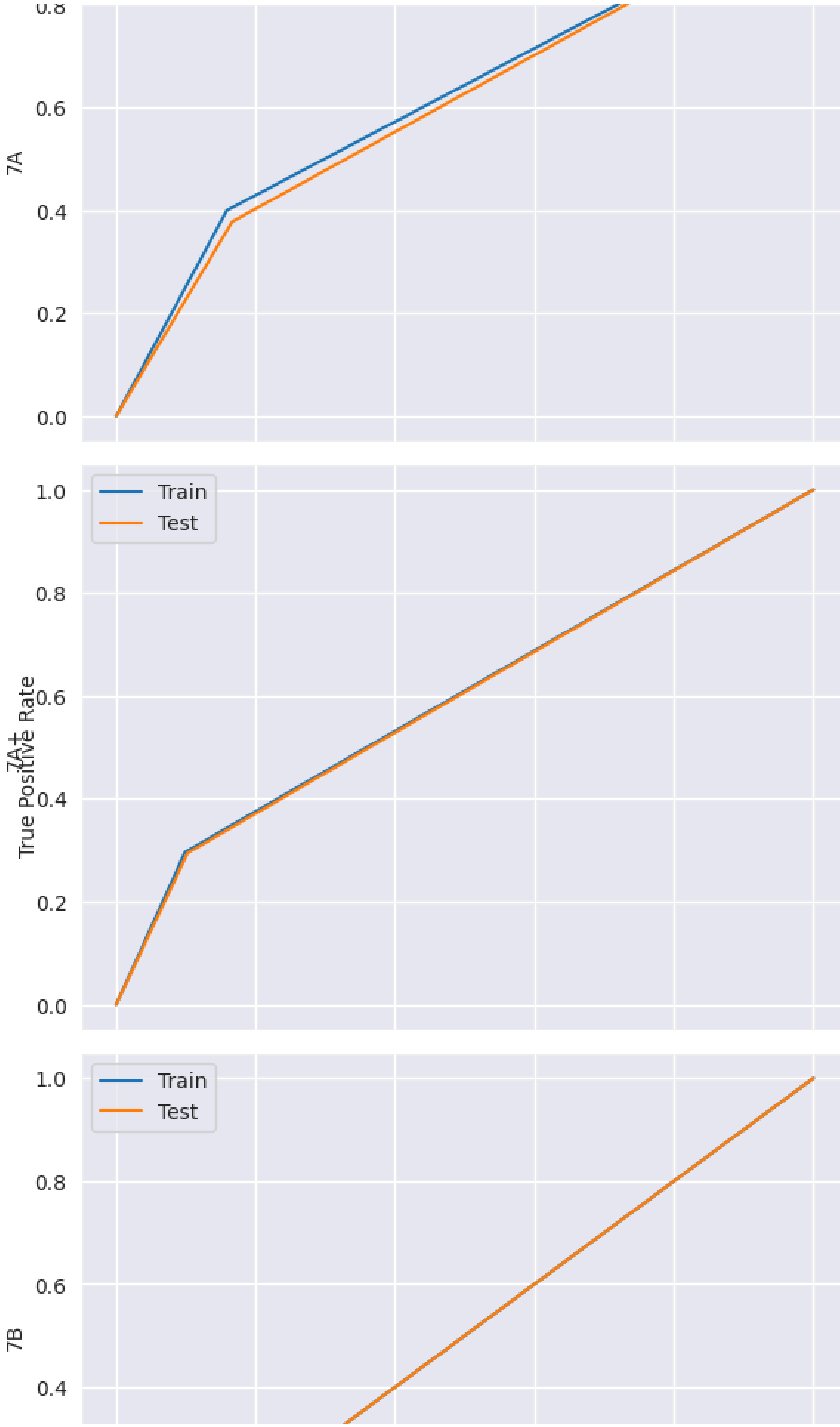
```
one_vs_rest_roc_curve(train_predicted, test_predicted)
```


Train
Test
1.0
0.8
0.6
0.4
0.2
0.0
5+
Train
Test
1.0
0.8
0.6
0.4
0.2
0.0
6A
Train
Test
1.0
0.8

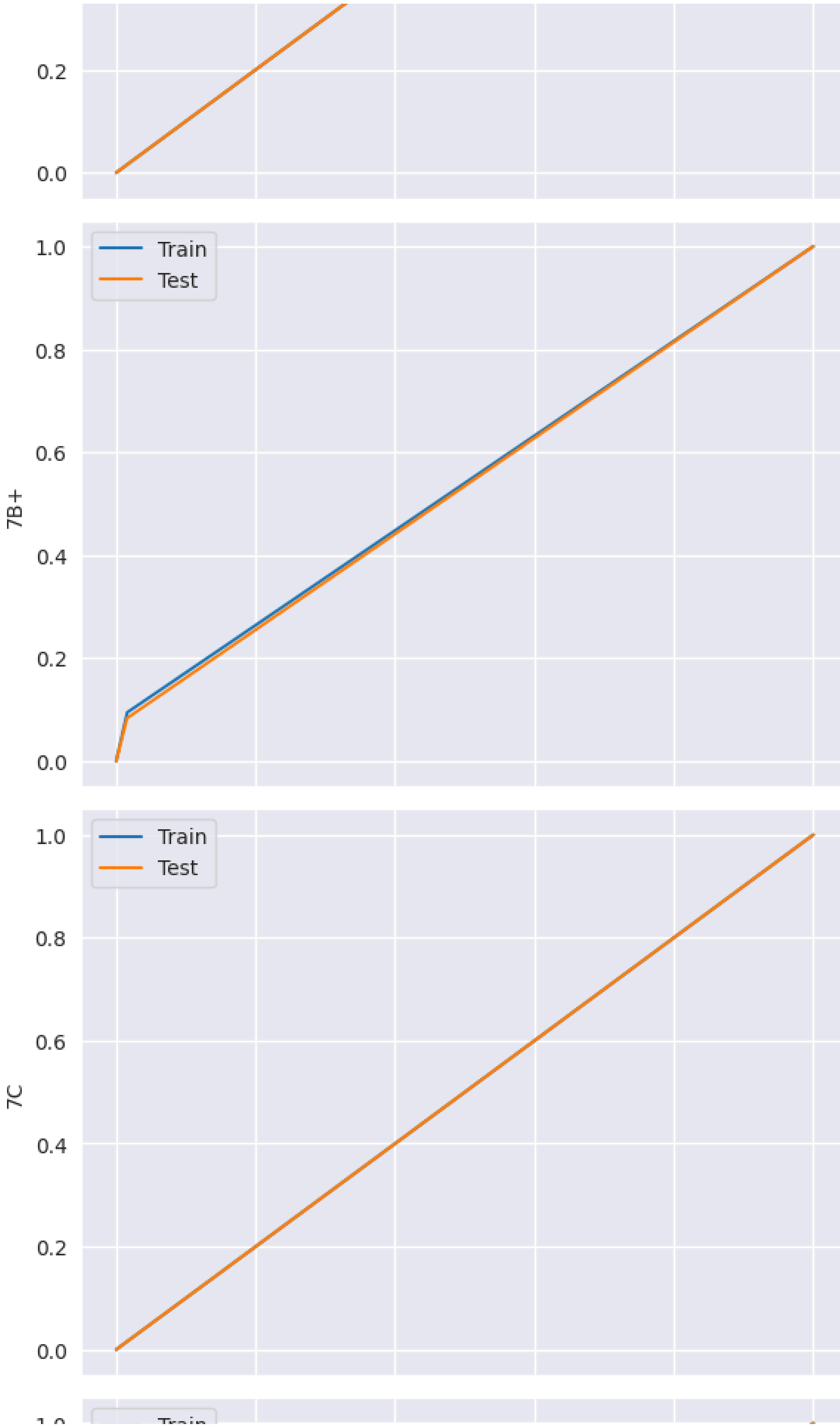


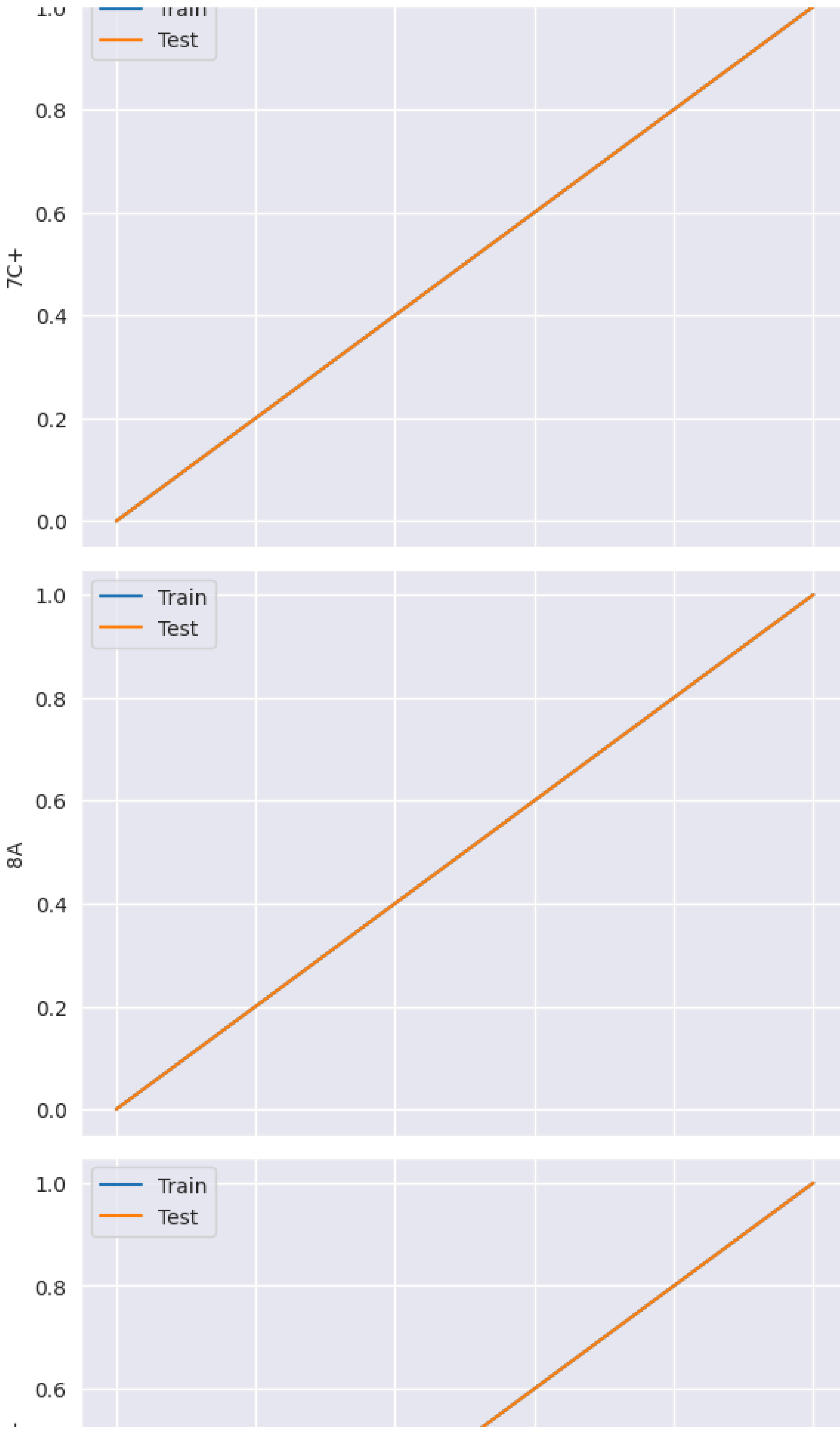
6A+
0.6
0.4
0.2
0.0
6B
Train
Test
1.0
0.8
0.6
0.4
0.2
0.0
6B+
Train
Test
1.0
0.8
0.6
0.4
0.2


0.0
Train
Test
1.0
0.8
0.6
6C
0.4
0.2
0.0
Train
Test
1.0
0.8
0.6
6C+
0.4
0.2
0.0
Train
Test
1.0

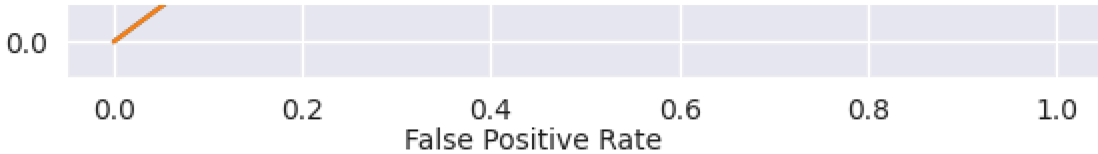


7A
0.8
0.6
0.4
0.2
0.0
7A+
True Positive Rate
Train
Test
1.0
0.8
0.6
0.4
0.2
0.0
7B
Train
Test
1.0
0.8
0.6
0.4


0.2
0.0
Train
Test
1.0
0.8
0.6
7B+
0.4
0.2
0.0
Train
Test
1.0
0.8
0.6
7C
0.4
0.2
0.0


Train
Test
1.0
0.8
0.6
0.4
0.2
0.0
7C+
8A

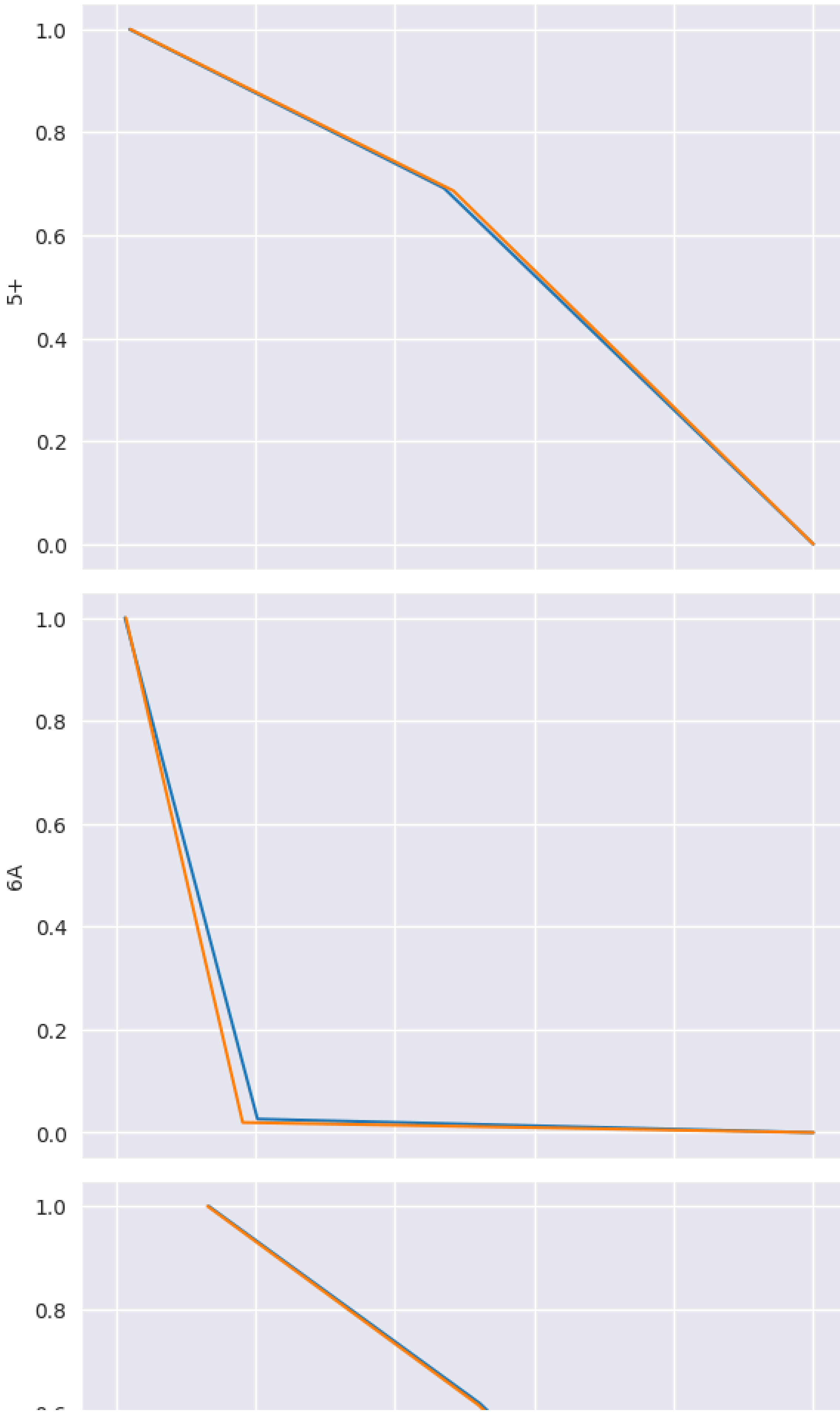
8A+
0.4
0.2
0.0
Train
Test
1.0
0.8
0.6
8B
0.4
0.2
0.0
Train
Test
1.0
0.8
0.6
8B+
0.4
0.2



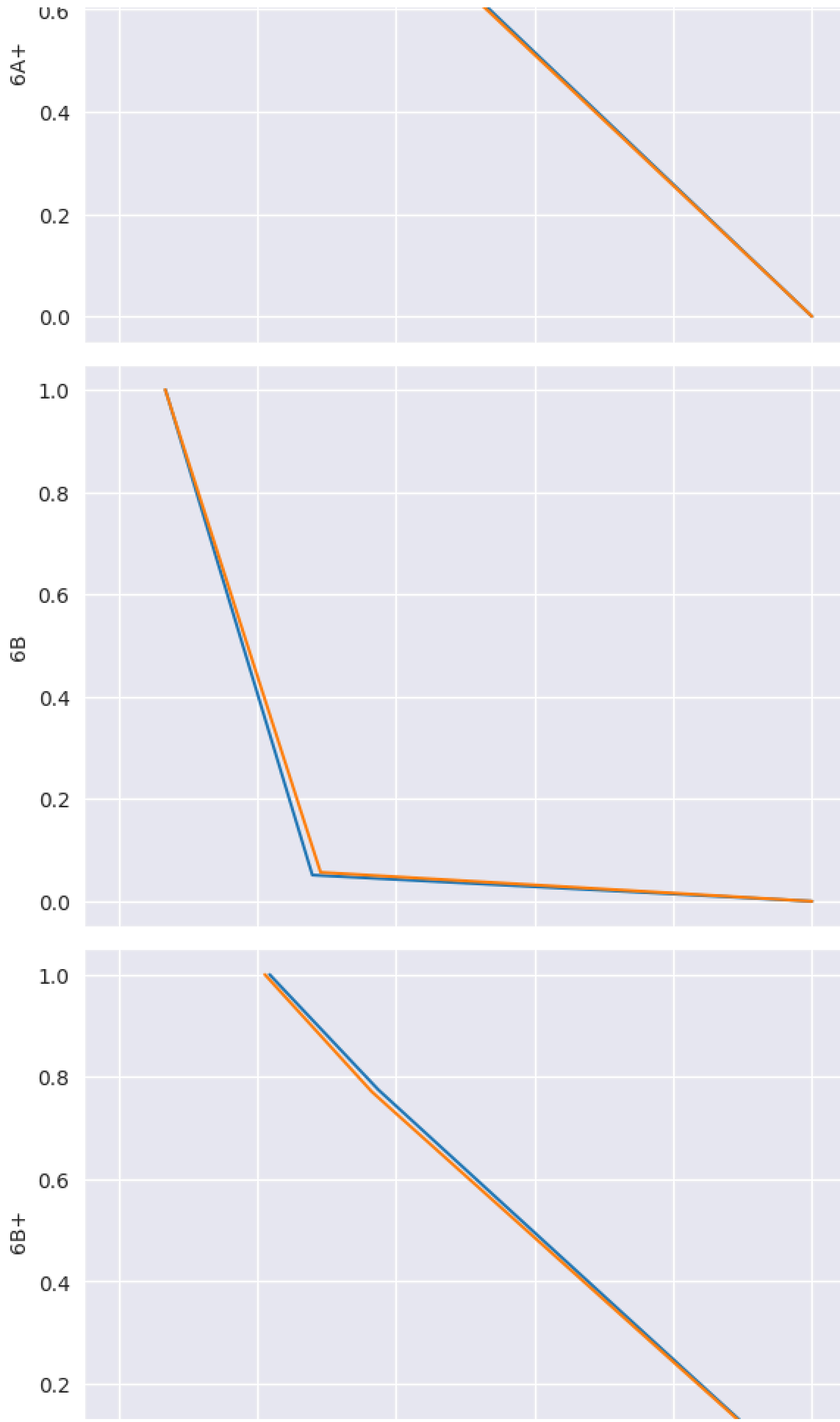
For almost every grade, this baseline model doesn't achieve better performance than a random guess. At least, it's not worse.

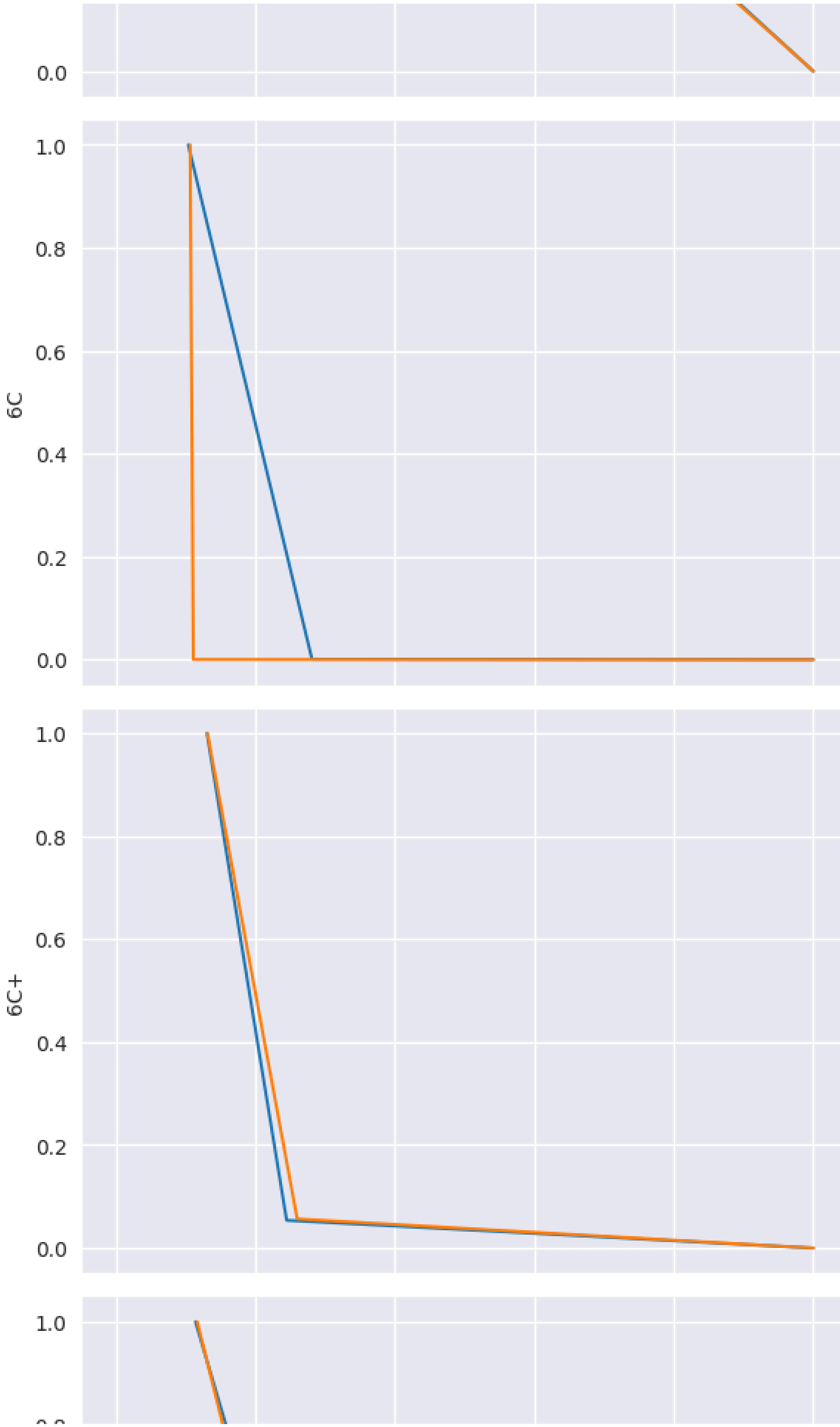
For middle grades (around 6), we get the same insight as with the confusion matrix: the model achieves better performance. Thus the class imbalance has a clear impact and we must apply corrections before going further.

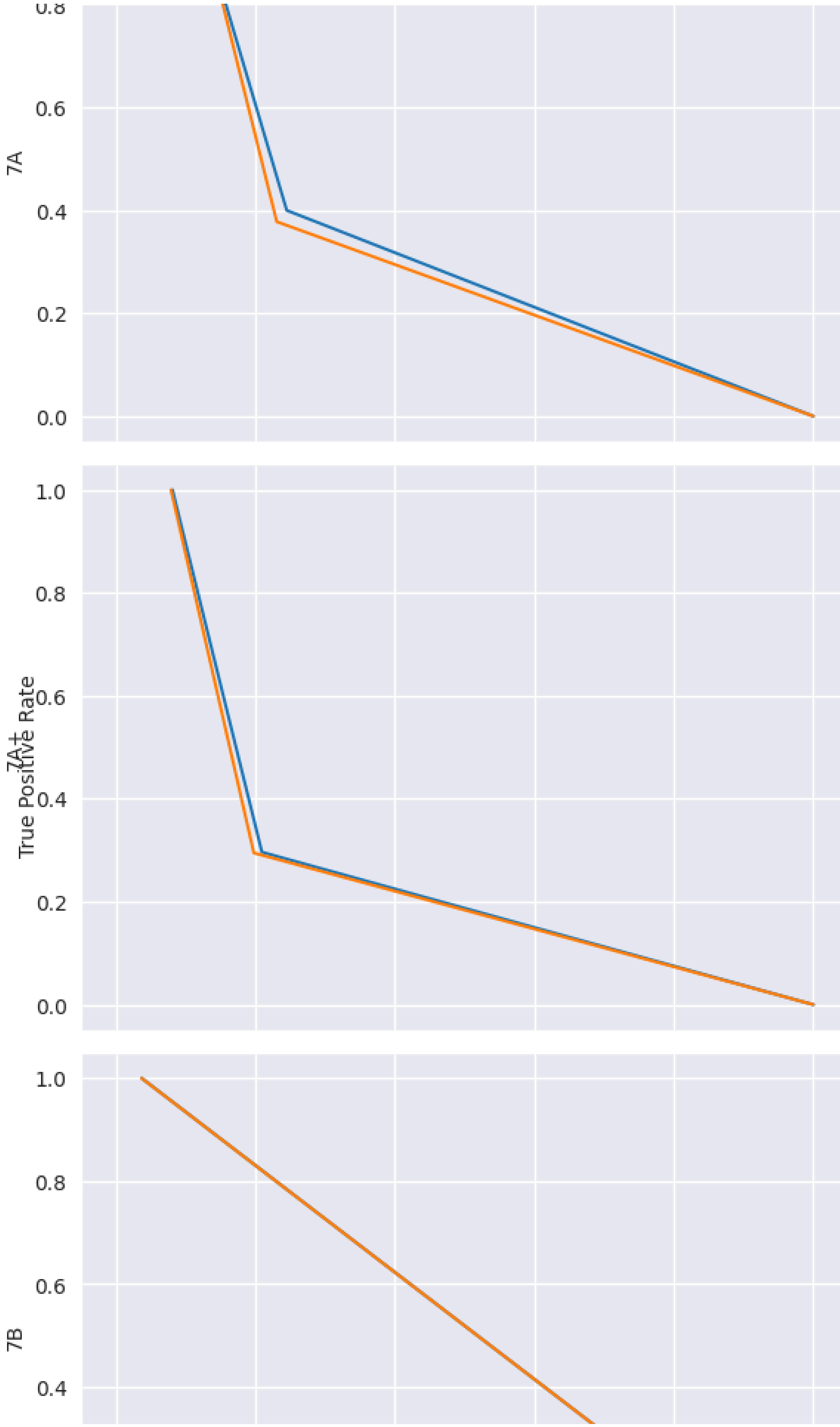
In [53]:
```
one_vs_rest_precision_recall_curve(train_predicted, test_predicted)
```



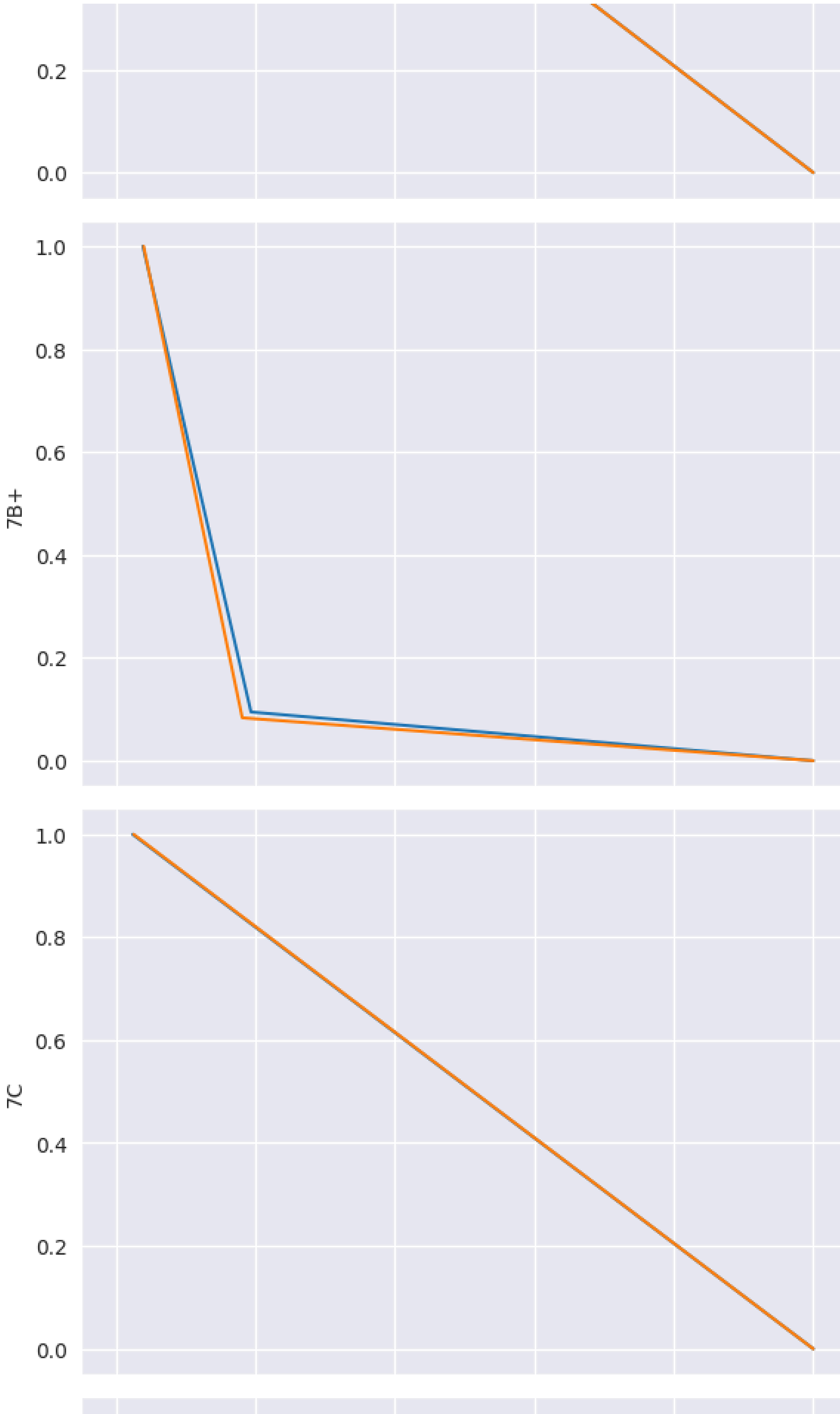
5+
6A
1.0
0.8
0.6
0.4
0.2
0.0

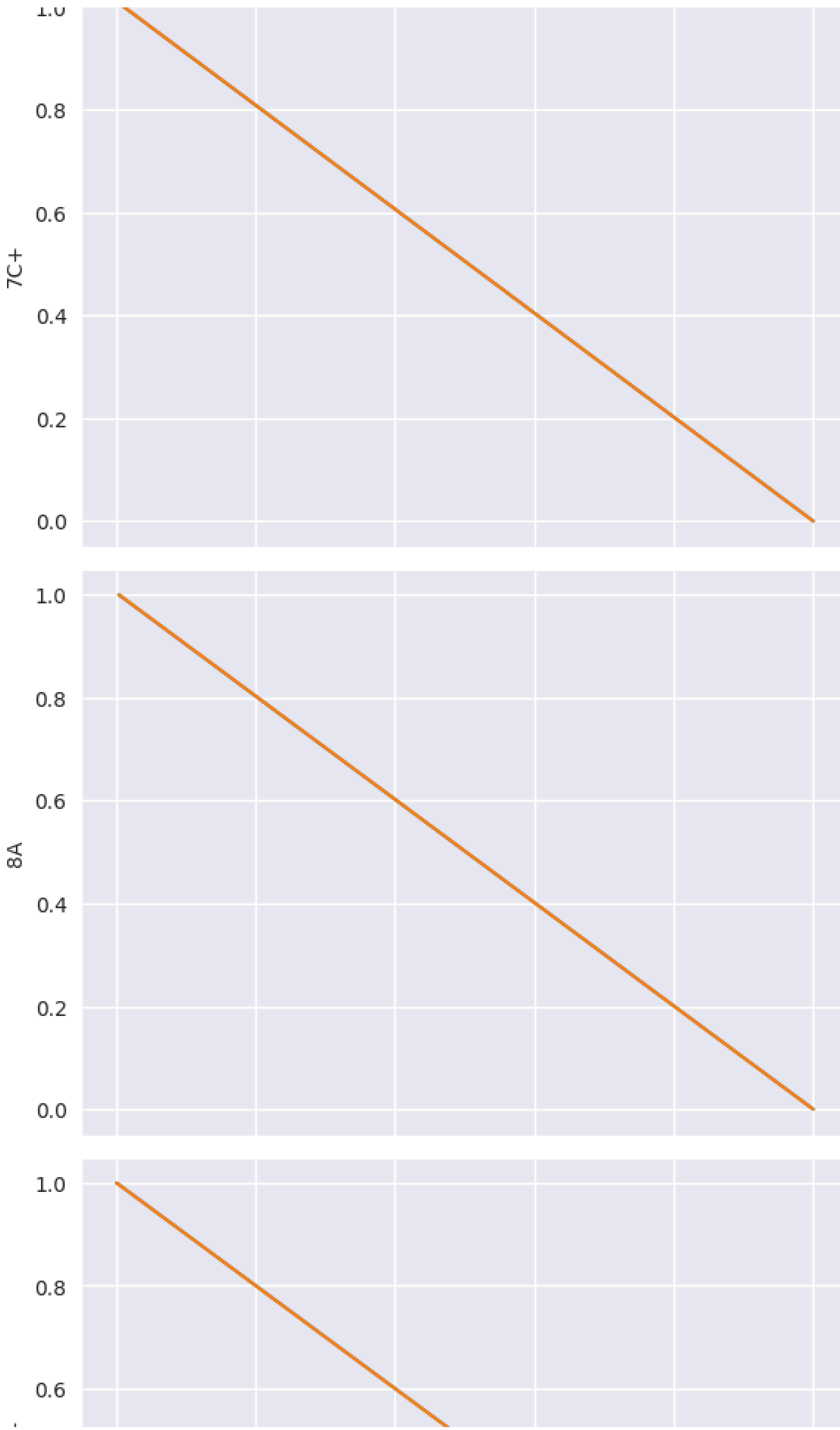

6A+
0.6
0.4
0.2
0.0
6B
1.0
0.8
0.6
0.4
0.2
0.0
6B+
1.0
0.8
0.6
0.4
0.2


0.0
1.0
0.8
0.6
0.4
0.2
0.0
6C
1.0
0.8
0.6
0.4
0.2
0.0
6C+
1.0

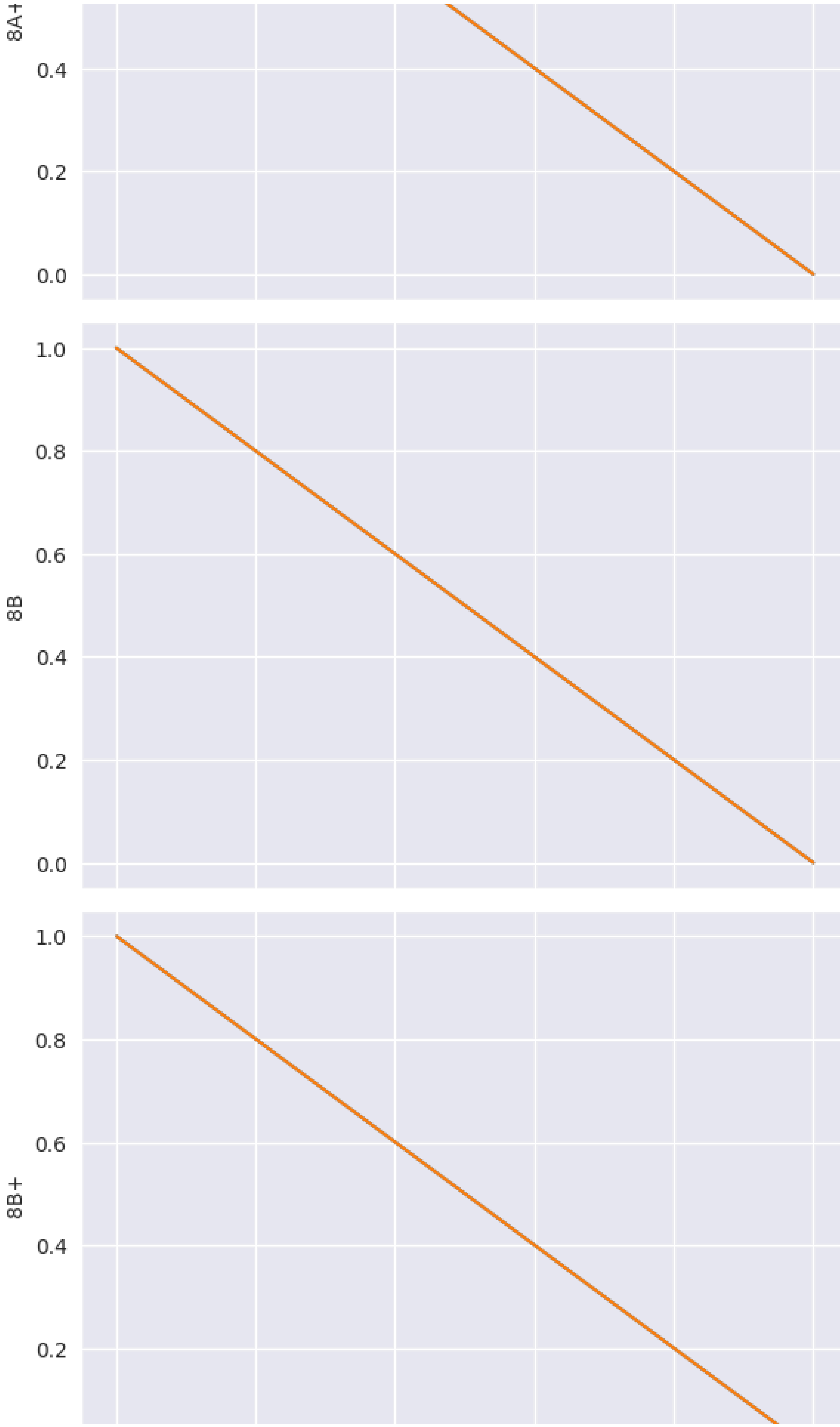


7A
0.8
0.6
0.4
0.2
0.0
7A+
True Positive Rate
1.0
0.8
0.6
0.4
0.2
0.0
7B
1.0
0.8
0.6
0.4


0.2
0.0
1.0
0.8
0.6
7B+
0.4
0.2
0.0
1.0
0.8
0.6
7C
0.4
0.2
0.0



1.0
0.8
0.6
7C+
0.4
0.2
0.0
1.0
0.8
0.6
8A
0.4
0.2
0.0
1.0
0.8
0.6



8A+
0.4
0.2
0.0
1.0
0.8
0.6
8B
0.4
0.2
0.0
1.0
0.8
0.6
8B+
0.4
0.2

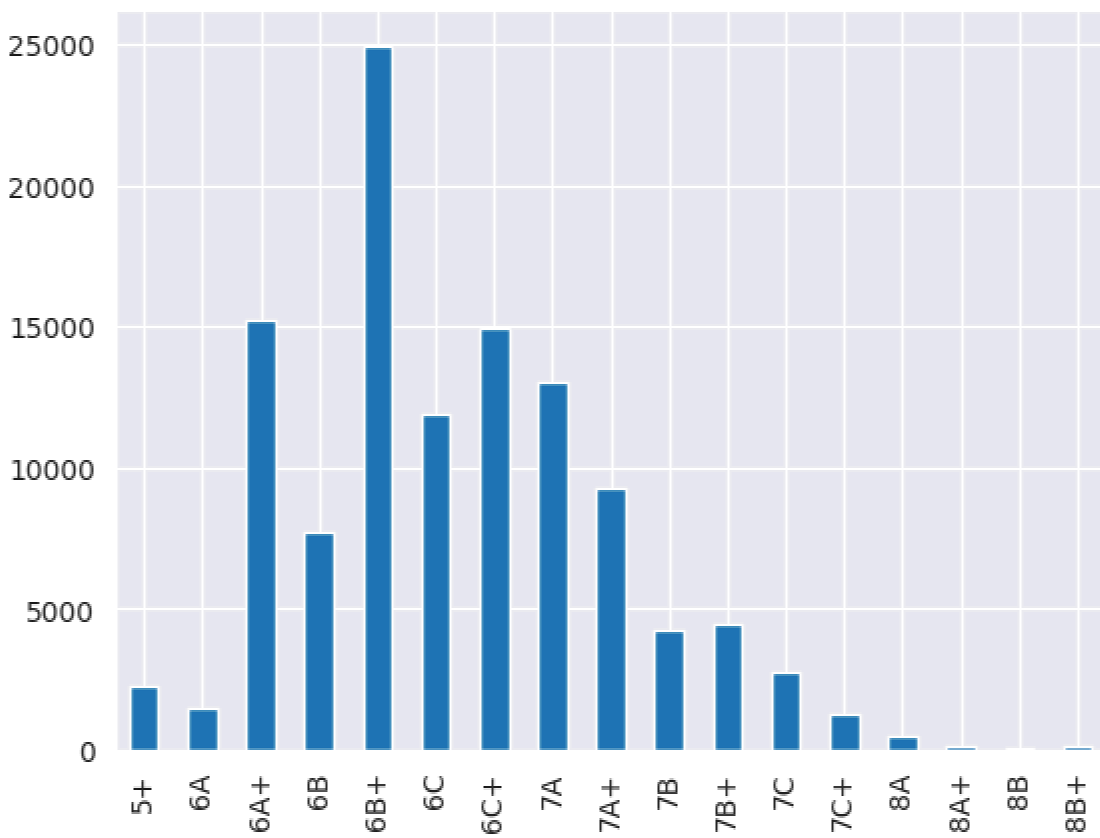
Class weights

We'll use a Keras feature called class weights, that allow our model to take into account class imbalances and adapt its learning.

Here is the distribution of classes in the training dataset:

In [63]:
```
train_labels.sum().plot(kind='bar')
```

Out[63]: <Axes: >



Calculate class weights

In [143…
```
from sklearn.utils.class_weight import compute_class_weight

class_weights = compute_class_weight(class_weight='balanced', classes=np.ara
class_weight_dict = dict(enumerate(class_weights))
```

Train model

In [80]:
```
train_model(
  model=compile_model(build_function=create_baseline),
  name='baseline_weighted',
  training_features=[train_moves, train_features],
  training_labels=train_labels,
  class_weight=class_weight_dict,
  epochs=50,
)
```

Model: "functional_3"

Layer (type)	**Output Shape**	**Param #**	**Connected to**
moves (InputLayer)	(None, 11, 18, 3)	0	-
flatten_3 (Flatten)	(None, 594)	0	moves[0][0]
features (InputLayer)	(None, 14)	0	-
concatenate_3 (Concatenate)	(None, 608)	0	flatten_3[0][features[0][0
dense_9 (Dense)	(None, 256)	155,904	concatenate_3
dense_10 (Dense)	(None, 64)	16,448	dense_9[0][0]
dropout_3 (Dropout)	(None, 64)	0	dense_10[0][0
dense_11 (Dense)	(None, 17)	1,105	dropout_3[0][

Total params: 173,457 (677.57 KB)

Trainable params: 173,457 (677.57 KB)

Non-trainable params: 0 (0.00 B)

```
None
Epoch 1/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.2428 - accuracy_at_
five: 0.6729 - accuracy_at_three: 0.5014 - loss: 2.5353 - val_accuracy: 0.14
46 - val_accuracy_at_five: 0.6215 - val_accuracy_at_three: 0.3960 - val_los
s: 2.4767
Epoch 2/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - accuracy: 0.3109 - accuracy_at
_five: 0.8159 - accuracy_at_three: 0.6287 - loss: 2.3743 - val_accuracy: 0.1
447 - val_accuracy_at_five: 0.6113 - val_accuracy_at_three: 0.3931 - val_los
s: 2.5119
Epoch 3/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.3159 - accuracy_at_
five: 0.8324 - accuracy_at_three: 0.6467 - loss: 2.4727 - val_accuracy: 0.11
80 - val_accuracy_at_five: 0.5079 - val_accuracy_at_three: 0.3266 - val_los
s: 2.8408
Epoch 4/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.3139 - accuracy_at_
five: 0.8300 - accuracy_at_three: 0.6476 - loss: 2.6298 - val_accuracy: 0.14
50 - val_accuracy_at_five: 0.6008 - val_accuracy_at_three: 0.3941 - val_los
s: 2.7189
Epoch 5/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - accuracy: 0.3083 - accuracy_at
_five: 0.8266 - accuracy_at_three: 0.6435 - loss: 2.6952 - val_accuracy: 0.1
434 - val_accuracy_at_five: 0.6025 - val_accuracy_at_three: 0.3914 - val_los
s: 2.7891
Epoch 6/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 6s 5ms/step - accuracy: 0.3095 - accuracy_at_
five: 0.8309 - accuracy_at_three: 0.6490 - loss: 2.7290 - val_accuracy: 0.14
78 - val_accuracy_at_five: 0.5962 - val_accuracy_at_three: 0.3912 - val_los
s: 2.7944
Epoch 7/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 11s 5ms/step - accuracy: 0.3115 - accuracy_at
_five: 0.8298 - accuracy_at_three: 0.6505 - loss: 2.6178 - val_accuracy: 0.1
474 - val_accuracy_at_five: 0.5844 - val_accuracy_at_three: 0.3870 - val_los
s: 2.9449
Epoch 8/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.3077 - accuracy_at_
five: 0.8262 - accuracy_at_three: 0.6449 - loss: 2.8302 - val_accuracy: 0.15
37 - val_accuracy_at_five: 0.6428 - val_accuracy_at_three: 0.4127 - val_los
s: 2.7454
Epoch 9/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.3092 - accuracy_at_
five: 0.8326 - accuracy_at_three: 0.6530 - loss: 2.7051 - val_accuracy: 0.14
98 - val_accuracy_at_five: 0.6131 - val_accuracy_at_three: 0.3939 - val_los
s: 3.0224
Epoch 10/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.3154 - accuracy_at_
five: 0.8335 - accuracy_at_three: 0.6530 - loss: 2.6540 - val_accuracy: 0.15
80 - val_accuracy_at_five: 0.6540 - val_accuracy_at_three: 0.4286 - val_los
s: 3.0453
Epoch 11/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.3123 - accuracy_at_
five: 0.8328 - accuracy_at_three: 0.6484 - loss: 2.8510 - val_accuracy: 0.15
47 - val_accuracy_at_five: 0.6701 - val_accuracy_at_three: 0.4333 - val_los
s: 3.0683
```

```
Epoch 12/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.3097 - accuracy_at_
five: 0.8271 - accuracy_at_three: 0.6469 - loss: 2.9088 - val_accuracy: 0.16
02 - val_accuracy_at_five: 0.6686 - val_accuracy_at_three: 0.4375 - val_los
s: 3.2841
Epoch 13/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.3126 - accuracy_at_
five: 0.8355 - accuracy_at_three: 0.6536 - loss: 2.8908 - val_accuracy: 0.14
69 - val_accuracy_at_five: 0.6344 - val_accuracy_at_three: 0.4116 - val_los
s: 3.0190
Epoch 14/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 10s 7ms/step - accuracy: 0.3160 - accuracy_at
_five: 0.8345 - accuracy_at_three: 0.6537 - loss: 2.8196 - val_accuracy: 0.1
394 - val_accuracy_at_five: 0.6303 - val_accuracy_at_three: 0.4030 - val_los
s: 3.0899
Epoch 15/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 10s 7ms/step - accuracy: 0.3076 - accuracy_at
_five: 0.8344 - accuracy_at_three: 0.6465 - loss: 3.1930 - val_accuracy: 0.1
441 - val_accuracy_at_five: 0.6451 - val_accuracy_at_three: 0.4107 - val_los
s: 3.3221
Epoch 16/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 9s 6ms/step - accuracy: 0.3085 - accuracy_at_
five: 0.8301 - accuracy_at_three: 0.6455 - loss: 2.9534 - val_accuracy: 0.14
69 - val_accuracy_at_five: 0.6604 - val_accuracy_at_three: 0.4194 - val_los
s: 3.8720
Epoch 17/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.3013 - accuracy_at_
five: 0.8258 - accuracy_at_three: 0.6394 - loss: 3.2115 - val_accuracy: 0.16
22 - val_accuracy_at_five: 0.6683 - val_accuracy_at_three: 0.4463 - val_los
s: 3.4717
Epoch 18/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - accuracy: 0.3048 - accuracy_at
_five: 0.8214 - accuracy_at_three: 0.6386 - loss: 3.4539 - val_accuracy: 0.1
479 - val_accuracy_at_five: 0.6511 - val_accuracy_at_three: 0.4173 - val_los
s: 3.5090
Epoch 19/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.3122 - accuracy_at_
five: 0.8337 - accuracy_at_three: 0.6499 - loss: 3.1065 - val_accuracy: 0.14
43 - val_accuracy_at_five: 0.6633 - val_accuracy_at_three: 0.4245 - val_los
s: 3.6486
Epoch 20/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.3075 - accuracy_at_
five: 0.8369 - accuracy_at_three: 0.6511 - loss: 2.9993 - val_accuracy: 0.13
56 - val_accuracy_at_five: 0.6256 - val_accuracy_at_three: 0.3934 - val_los
s: 3.6508
Epoch 21/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.3050 - accuracy_at_
five: 0.8300 - accuracy_at_three: 0.6425 - loss: 3.3489 - val_accuracy: 0.13
52 - val_accuracy_at_five: 0.6249 - val_accuracy_at_three: 0.3968 - val_los
s: 3.6111
Epoch 22/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.3016 - accuracy_at_
five: 0.8242 - accuracy_at_three: 0.6373 - loss: 3.5063 - val_accuracy: 0.15
10 - val_accuracy_at_five: 0.6704 - val_accuracy_at_three: 0.4434 - val_los
s: 3.7648
Epoch 23/50
```

```
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.3036 - accuracy_at_
five: 0.8261 - accuracy_at_three: 0.6384 - loss: 3.2866 - val_accuracy: 0.15
39 - val_accuracy_at_five: 0.6723 - val_accuracy_at_three: 0.4398 - val_los
s: 3.8299
Epoch 24/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.3068 - accuracy_at_
five: 0.8272 - accuracy_at_three: 0.6373 - loss: 3.2895 - val_accuracy: 0.14
92 - val_accuracy_at_five: 0.6789 - val_accuracy_at_three: 0.4452 - val_los
s: 4.5273
Epoch 25/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.3073 - accuracy_at_
five: 0.8244 - accuracy_at_three: 0.6410 - loss: 3.8106 - val_accuracy: 0.14
78 - val_accuracy_at_five: 0.6587 - val_accuracy_at_three: 0.4313 - val_los
s: 4.1668
Epoch 26/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.3019 - accuracy_at_
five: 0.8233 - accuracy_at_three: 0.6376 - loss: 3.6599 - val_accuracy: 0.14
94 - val_accuracy_at_five: 0.6640 - val_accuracy_at_three: 0.4333 - val_los
s: 4.4717
Epoch 27/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.3113 - accuracy_at_
five: 0.8314 - accuracy_at_three: 0.6435 - loss: 3.6933 - val_accuracy: 0.16
08 - val_accuracy_at_five: 0.6916 - val_accuracy_at_three: 0.4630 - val_los
s: 4.7114
Epoch 28/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.3096 - accuracy_at_
five: 0.8260 - accuracy_at_three: 0.6381 - loss: 3.1507 - val_accuracy: 0.16
56 - val_accuracy_at_five: 0.6847 - val_accuracy_at_three: 0.4568 - val_los
s: 4.6378
Epoch 29/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.3088 - accuracy_at_
five: 0.8255 - accuracy_at_three: 0.6421 - loss: 3.5571 - val_accuracy: 0.15
64 - val_accuracy_at_five: 0.6786 - val_accuracy_at_three: 0.4448 - val_los
s: 4.9956
Epoch 30/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.3116 - accuracy_at_
five: 0.8257 - accuracy_at_three: 0.6386 - loss: 3.4877 - val_accuracy: 0.16
03 - val_accuracy_at_five: 0.6737 - val_accuracy_at_three: 0.4491 - val_los
s: 5.4028
Epoch 31/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.3101 - accuracy_at_
five: 0.8261 - accuracy_at_three: 0.6426 - loss: 3.5341 - val_accuracy: 0.15
16 - val_accuracy_at_five: 0.6724 - val_accuracy_at_three: 0.4378 - val_los
s: 5.1260
Epoch 32/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 11s 6ms/step - accuracy: 0.3029 - accuracy_at
_five: 0.8123 - accuracy_at_three: 0.6303 - loss: 4.4725 - val_accuracy: 0.1
577 - val_accuracy_at_five: 0.6727 - val_accuracy_at_three: 0.4399 - val_los
s: 5.5916
Epoch 33/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.3093 - accuracy_at_
five: 0.8252 - accuracy_at_three: 0.6388 - loss: 3.2874 - val_accuracy: 0.15
48 - val_accuracy_at_five: 0.6700 - val_accuracy_at_three: 0.4338 - val_los
s: 5.4035
Epoch 34/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.3070 - accuracy_at_
```

```
five: 0.8234 - accuracy_at_three: 0.6380 - loss: 3.8493 - val_accuracy: 0.14
89 - val_accuracy_at_five: 0.6455 - val_accuracy_at_three: 0.4116 - val_los
s: 5.4015
Epoch 35/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 9s 4ms/step - accuracy: 0.3070 - accuracy_at_
five: 0.8127 - accuracy_at_three: 0.6276 - loss: 4.1267 - val_accuracy: 0.15
96 - val_accuracy_at_five: 0.6619 - val_accuracy_at_three: 0.4386 - val_los
s: 5.7758
Epoch 36/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 11s 5ms/step - accuracy: 0.3095 - accuracy_at
_five: 0.8187 - accuracy_at_three: 0.6359 - loss: 3.6403 - val_accuracy: 0.1
553 - val_accuracy_at_five: 0.6627 - val_accuracy_at_three: 0.4292 - val_los
s: 5.9399
Epoch 37/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.3059 - accuracy_at_
five: 0.8199 - accuracy_at_three: 0.6348 - loss: 3.9222 - val_accuracy: 0.14
16 - val_accuracy_at_five: 0.6335 - val_accuracy_at_three: 0.4037 - val_los
s: 5.8814
Epoch 38/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.3078 - accuracy_at_
five: 0.8179 - accuracy_at_three: 0.6378 - loss: 3.3678 - val_accuracy: 0.14
47 - val_accuracy_at_five: 0.6418 - val_accuracy_at_three: 0.4142 - val_los
s: 6.6404
Epoch 39/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.3095 - accuracy_at_
five: 0.8196 - accuracy_at_three: 0.6345 - loss: 3.5486 - val_accuracy: 0.14
43 - val_accuracy_at_five: 0.6311 - val_accuracy_at_three: 0.4031 - val_los
s: 6.5066
Epoch 40/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.2994 - accuracy_at_
five: 0.7995 - accuracy_at_three: 0.6182 - loss: 4.6639 - val_accuracy: 0.14
72 - val_accuracy_at_five: 0.6454 - val_accuracy_at_three: 0.4101 - val_los
s: 6.9660
Epoch 41/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.3035 - accuracy_at_
five: 0.8062 - accuracy_at_three: 0.6249 - loss: 4.7300 - val_accuracy: 0.14
85 - val_accuracy_at_five: 0.6466 - val_accuracy_at_three: 0.4171 - val_los
s: 7.2524
Epoch 42/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.3038 - accuracy_at_
five: 0.8074 - accuracy_at_three: 0.6237 - loss: 4.1372 - val_accuracy: 0.13
98 - val_accuracy_at_five: 0.6205 - val_accuracy_at_three: 0.3922 - val_los
s: 7.2662
Epoch 43/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.3042 - accuracy_at_
five: 0.8110 - accuracy_at_three: 0.6249 - loss: 4.5972 - val_accuracy: 0.15
00 - val_accuracy_at_five: 0.6461 - val_accuracy_at_three: 0.4187 - val_los
s: 7.7529
Epoch 44/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 11s 5ms/step - accuracy: 0.3076 - accuracy_at
_five: 0.8098 - accuracy_at_three: 0.6259 - loss: 3.9564 - val_accuracy: 0.1
448 - val_accuracy_at_five: 0.6356 - val_accuracy_at_three: 0.4074 - val_los
s: 7.9256
Epoch 45/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 11s 5ms/step - accuracy: 0.3070 - accuracy_at
_five: 0.8103 - accuracy_at_three: 0.6235 - loss: 4.4248 - val_accuracy: 0.1
```

```
487 - val_accuracy_at_five: 0.6474 - val_accuracy_at_three: 0.4149 - val_los
s: 8.2090
Epoch 46/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 9s 4ms/step - accuracy: 0.2988 - accuracy_at_
five: 0.7977 - accuracy_at_three: 0.6165 - loss: 4.5495 - val_accuracy: 0.14
14 - val_accuracy_at_five: 0.6403 - val_accuracy_at_three: 0.4089 - val_los
s: 8.6875
Epoch 47/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.3017 - accuracy_at_
five: 0.8026 - accuracy_at_three: 0.6183 - loss: 4.2116 - val_accuracy: 0.14
97 - val_accuracy_at_five: 0.6382 - val_accuracy_at_three: 0.4130 - val_los
s: 8.3520
Epoch 48/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 11s 5ms/step - accuracy: 0.3061 - accuracy_at
_five: 0.8123 - accuracy_at_three: 0.6270 - loss: 6.2940 - val_accuracy: 0.1
491 - val_accuracy_at_five: 0.6571 - val_accuracy_at_three: 0.4264 - val_los
s: 8.5736
Epoch 49/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.3020 - accuracy_at_
five: 0.8077 - accuracy_at_three: 0.6215 - loss: 4.2796 - val_accuracy: 0.15
42 - val_accuracy_at_five: 0.6603 - val_accuracy_at_three: 0.4310 - val_los
s: 9.6011
Epoch 50/50
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.2983 - accuracy_at_
five: 0.8069 - accuracy_at_three: 0.6190 - loss: 5.0088 - val_accuracy: 0.14
24 - val_accuracy_at_five: 0.6332 - val_accuracy_at_three: 0.4096 - val_los
s: 9.1847
```

Out[80]:

```
<Functional name=functional_3, built=True>
```

This weighted training made the model overfit farther than before: the model started to learn some patterns.

Accuracies

In [86]:

```
baseline_weighted, train_predicted, train_y_true, train_y_pred, test_predict
```

In [87]:

```
print_accuracies(baseline_weighted, test_y_true, test_y_pred)
```

```
Accuracy: 28.42%
Balanced Accuracy: 24.04%
Accuracy for Top3: 58.58%
Accuracy for Top5: 78.82%
```

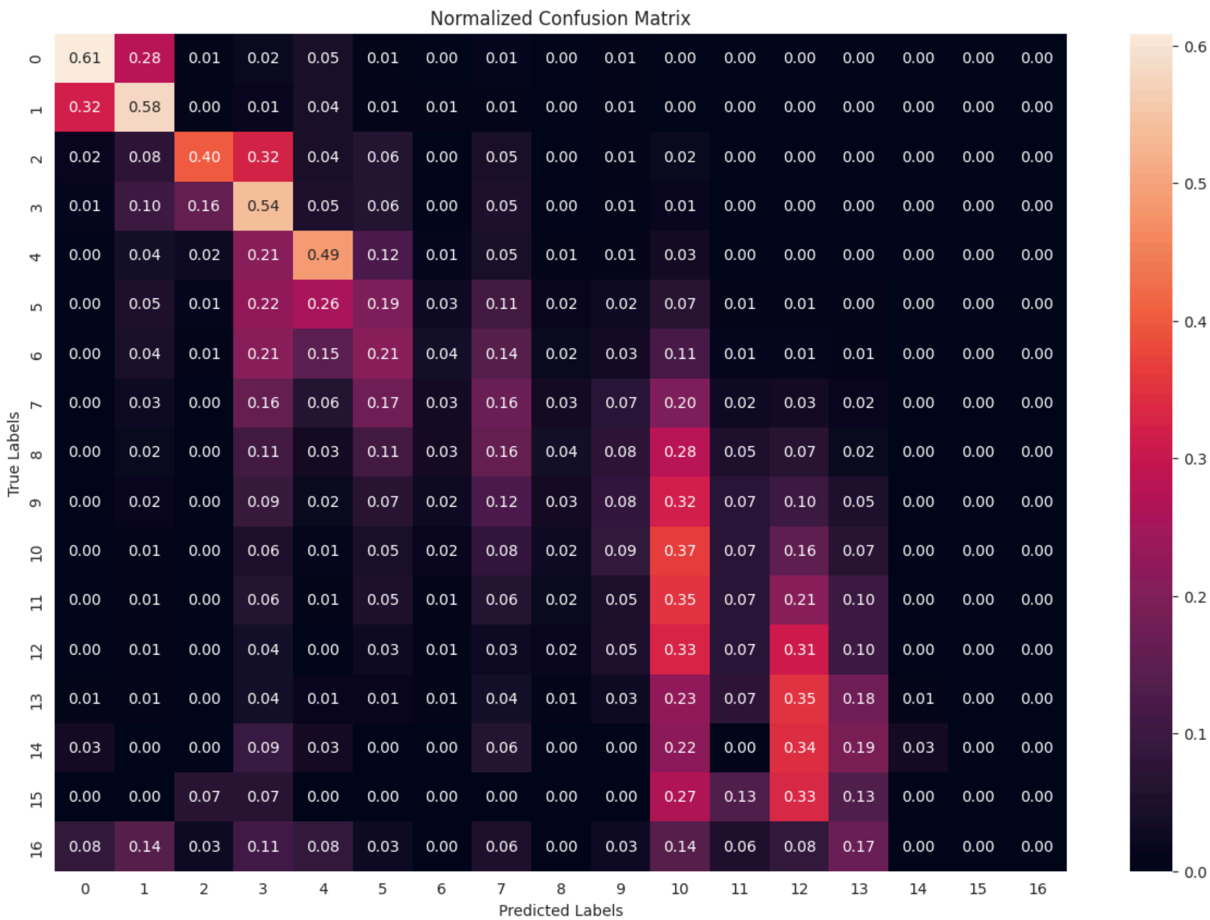
Global accuracies are less important, because they are too precise for our climbing problem.

Here, the balanced accuracy is better, which is a sign of our model generalizing more and not only skipping rare grades.

Confusion matrix

In [88]:
```
confusion_matrix_analysis(test_y_true, test_y_pred)
```

```
Accuracy for grade 5+: 60.88%
Accuracy for grade 6A: 57.95%
Accuracy for grade 6A+: 39.95%
Accuracy for grade 6B: 53.89%
Accuracy for grade 6B+: 48.70%
Accuracy for grade 6C: 19.42%
Accuracy for grade 6C+: 3.64%
Accuracy for grade 7A: 16.46%
Accuracy for grade 7A+: 3.93%
Accuracy for grade 7B: 8.30%
Accuracy for grade 7B+: 36.79%
Accuracy for grade 7C: 6.97%
Accuracy for grade 7C+: 31.08%
Accuracy for grade 8A: 17.65%
Accuracy for grade 8A+: 3.12%
Accuracy for grade 8B: 0.00%
Accuracy for grade 8B+: 0.00%
```



Predictions for easier grades are more concentrated on the diagonal, and our model now predicts difficult grades.

The new problem here is that our model tends to overestimate difficult routes (lower triangle is more filled in the sub-matrix [10:, 10:])

Combine under-sampling and over-sampling

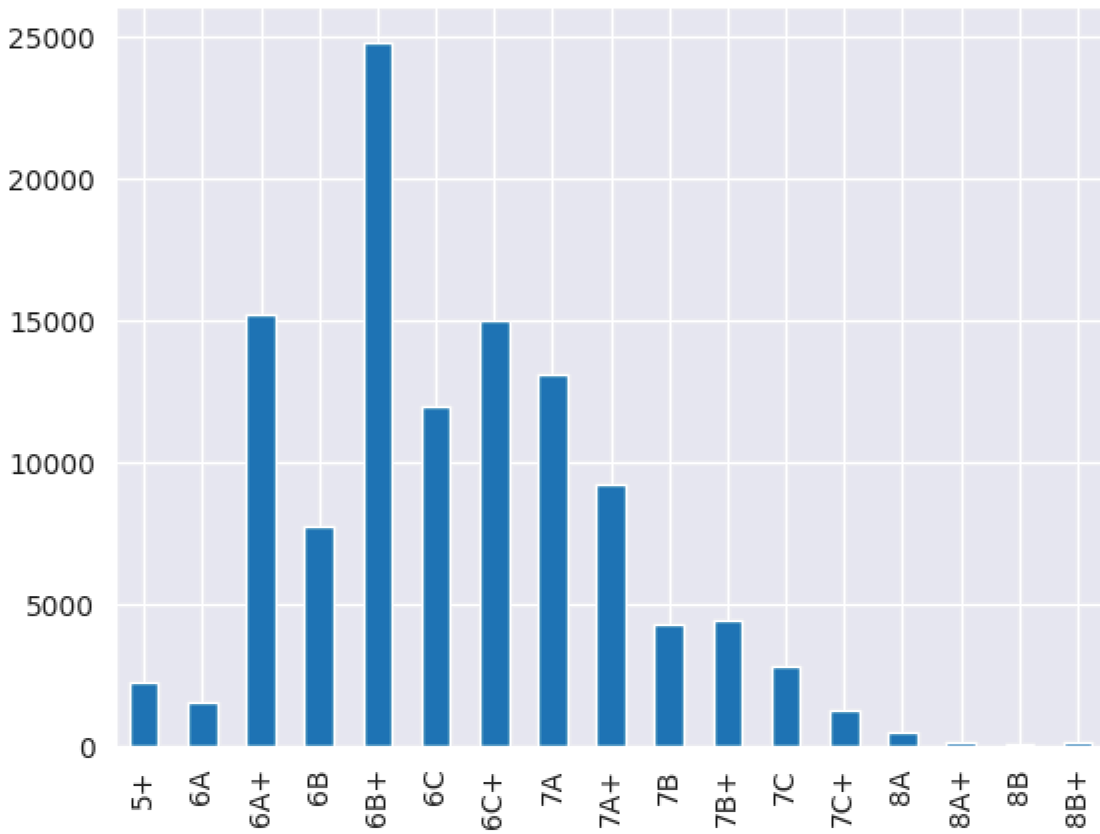
We'll try to keep all counts in range `[10000; 20000]` except for extreme classes (8A and further).

In [104…

```
train_labels.sum().plot(kind='bar')
```

Out[104…

```
<Axes: >
```



Under-sampling

In [147…

```
from imblearn.under_sampling import RandomUnderSampler

under_sampler = RandomUnderSampler(
  sampling_strategy={
    2: 10_000,
    4: 10_000,
    5: 10_000,
    6: 10_000,
    7: 10_000
  }
)
```

```
train_features_undersample, train_labels_undersample = under_sampler.fit_res
train_moves_undersample, _ = under_sampler.fit_resample(train_moves.reshape(
```

Over-sampling

In [150…

```
from imblearn.over_sampling import SMOTE

over_sampler = SMOTE(sampling_strategy={
  0: 10_000,
  1: 10_000,
  3: 10_000,
  8: 10_000,
  9: 10_000,
  10: 10_000,
  11: 10_000,
  12: 10_000,
  13: 5_000,
  14: 5_000,
  15: 5_000,
  16: 5_000,
})
train_combined_undersample = np.concatenate([train_features_undersample, tra

train_combined_resampled, train_labels_resampled = over_sampler.fit_resample
```

In [151…

```
train_features_resampled = train_combined_resampled[:, :nb_features]
train_moves_resampled = train_combined_resampled[:, nb_features:].reshape(-1
```

In [154…

```
train_labels_resampled.sum(axis=0)
```

Out[154…

```
array([10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
       10000, 10000, 10000, 10000,  5000,  5000,  5000,  5000])
```

Model training

In [155…

```
train_model(
  model=compile_model(build_function=create_baseline),
  name='baseline_resampled',
  training_features=[train_moves_resampled, train_features_resampled],
  training_labels=train_labels_resampled,
  epochs=50,
)
```

Model: "functional_5"

Layer (type)	Output Shape	Param #	Connected to
moves (InputLayer)	(None, 11, 18, 3)	0	-
flatten_5 (Flatten)	(None, 594)	0	moves[0][0]
features (InputLayer)	(None, 14)	0	-
concatenate_5 (Concatenate)	(None, 608)	0	flatten_5[0][features[0][0
dense_15 (Dense)	(None, 256)	155,904	concatenate_5
dense_16 (Dense)	(None, 64)	16,448	dense_15[0][0
dropout_5 (Dropout)	(None, 64)	0	dense_16[0][0
dense_17 (Dense)	(None, 17)	1,105	dropout_5[0][

Total params: 173,457 (677.57 KB)

Trainable params: 173,457 (677.57 KB)

Non-trainable params: 0 (0.00 B)

```
None
Epoch 1/50
```

```
2024-08-29 16:45:44.069606: W external/local_tsl/tsl/framework/cpu_allocator
_impl.cc:83] Allocation of 285120000 exceeds 10% of free system memory.
```

```
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 11s 5ms/step - accuracy: 0.2933 - accuracy_at
_five: 0.7772 - accuracy_at_three: 0.6065 - loss: 1.9541 - val_accuracy: 0.0
299 - val_accuracy_at_five: 0.1290 - val_accuracy_at_three: 0.0605 - val_los
s: 5.6243
Epoch 2/50
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.4371 - accuracy_at_
five: 0.9240 - accuracy_at_three: 0.7953 - loss: 1.4861 - val_accuracy: 0.05
62 - val_accuracy_at_five: 0.3259 - val_accuracy_at_three: 0.0665 - val_los
s: 5.4683
Epoch 3/50
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - accuracy: 0.4773 - accuracy_at
_five: 0.9426 - accuracy_at_three: 0.8310 - loss: 1.3752 - val_accuracy: 0.0
532 - val_accuracy_at_five: 0.3165 - val_accuracy_at_three: 0.0641 - val_los
s: 5.1612
Epoch 4/50
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.5023 - accuracy_at_
five: 0.9528 - accuracy_at_three: 0.8487 - loss: 1.3118 - val_accuracy: 0.05
31 - val_accuracy_at_five: 0.3373 - val_accuracy_at_three: 0.0660 - val_los
s: 5.4748
Epoch 5/50
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - accuracy: 0.5230 - accuracy_at
_five: 0.9579 - accuracy_at_three: 0.8582 - loss: 1.2652 - val_accuracy: 0.0
546 - val_accuracy_at_five: 0.3496 - val_accuracy_at_three: 0.0666 - val_los
s: 5.2508
Epoch 6/50
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.5357 - accuracy_at_
five: 0.9602 - accuracy_at_three: 0.8631 - loss: 1.2335 - val_accuracy: 0.05
71 - val_accuracy_at_five: 0.3694 - val_accuracy_at_three: 0.0808 - val_los
s: 5.1741
Epoch 7/50
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - accuracy: 0.5468 - accuracy_at
_five: 0.9623 - accuracy_at_three: 0.8726 - loss: 1.2049 - val_accuracy: 0.0
549 - val_accuracy_at_five: 0.3645 - val_accuracy_at_three: 0.0950 - val_los
s: 4.9856
Epoch 8/50
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - accuracy: 0.5586 - accuracy_at
_five: 0.9640 - accuracy_at_three: 0.8757 - loss: 1.1871 - val_accuracy: 0.0
571 - val_accuracy_at_five: 0.3783 - val_accuracy_at_three: 0.0971 - val_los
s: 5.4231
Epoch 9/50
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - accuracy: 0.5639 - accuracy_at
_five: 0.9659 - accuracy_at_three: 0.8786 - loss: 1.1765 - val_accuracy: 0.0
618 - val_accuracy_at_five: 0.3974 - val_accuracy_at_three: 0.1118 - val_los
s: 5.0594
Epoch 10/50
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 11s 6ms/step - accuracy: 0.5694 - accuracy_at
_five: 0.9661 - accuracy_at_three: 0.8824 - loss: 1.1602 - val_accuracy: 0.0
565 - val_accuracy_at_five: 0.3850 - val_accuracy_at_three: 0.1095 - val_los
s: 4.8509
Epoch 11/50
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 20s 5ms/step - accuracy: 0.5745 - accuracy_at
_five: 0.9666 - accuracy_at_three: 0.8830 - loss: 1.1524 - val_accuracy: 0.0
604 - val_accuracy_at_five: 0.3927 - val_accuracy_at_three: 0.1526 - val_los
s: 4.8085
Epoch 12/50
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.5793 - accuracy_at_
```

```
five: 0.9670 - accuracy_at_three: 0.8846 - loss: 1.1507 - val_accuracy: 0.05
79 - val_accuracy_at_five: 0.4057 - val_accuracy_at_three: 0.1668 - val_los
s: 4.6777
Epoch 13/50
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.5813 - accuracy_at_
five: 0.9679 - accuracy_at_three: 0.8866 - loss: 1.1393 - val_accuracy: 0.05
73 - val_accuracy_at_five: 0.4051 - val_accuracy_at_three: 0.1729 - val_los
s: 4.6814
Epoch 14/50
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - accuracy: 0.5833 - accuracy_at
_five: 0.9681 - accuracy_at_three: 0.8852 - loss: 1.1404 - val_accuracy: 0.0
570 - val_accuracy_at_five: 0.4059 - val_accuracy_at_three: 0.1689 - val_los
s: 4.6654
Epoch 15/50
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.5860 - accuracy_at_
five: 0.9672 - accuracy_at_three: 0.8861 - loss: 1.1355 - val_accuracy: 0.05
36 - val_accuracy_at_five: 0.3933 - val_accuracy_at_three: 0.1486 - val_los
s: 4.7210
Epoch 16/50
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.5871 - accuracy_at_
five: 0.9673 - accuracy_at_three: 0.8859 - loss: 1.1375 - val_accuracy: 0.05
73 - val_accuracy_at_five: 0.3987 - val_accuracy_at_three: 0.1781 - val_los
s: 4.8029
Epoch 17/50
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - accuracy: 0.5880 - accuracy_at
_five: 0.9671 - accuracy_at_three: 0.8851 - loss: 1.1373 - val_accuracy: 0.0
608 - val_accuracy_at_five: 0.4210 - val_accuracy_at_three: 0.1934 - val_los
s: 4.9696
Epoch 18/50
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - accuracy: 0.5900 - accuracy_at
_five: 0.9665 - accuracy_at_three: 0.8844 - loss: 1.1349 - val_accuracy: 0.0
572 - val_accuracy_at_five: 0.3959 - val_accuracy_at_three: 0.1584 - val_los
s: 5.0763
Epoch 19/50
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.5923 - accuracy_at_
five: 0.9671 - accuracy_at_three: 0.8851 - loss: 1.1379 - val_accuracy: 0.05
80 - val_accuracy_at_five: 0.4049 - val_accuracy_at_three: 0.1767 - val_los
s: 4.9590
Epoch 20/50
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.5937 - accuracy_at_
five: 0.9689 - accuracy_at_three: 0.8870 - loss: 1.1222 - val_accuracy: 0.05
42 - val_accuracy_at_five: 0.4086 - val_accuracy_at_three: 0.1812 - val_los
s: 4.7314
Epoch 21/50
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - accuracy: 0.5954 - accuracy_at
_five: 0.9667 - accuracy_at_three: 0.8865 - loss: 1.1271 - val_accuracy: 0.0
536 - val_accuracy_at_five: 0.3991 - val_accuracy_at_three: 0.1733 - val_los
s: 4.8786
Epoch 22/50
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - accuracy: 0.5950 - accuracy_at
_five: 0.9680 - accuracy_at_three: 0.8861 - loss: 1.1271 - val_accuracy: 0.0
591 - val_accuracy_at_five: 0.3993 - val_accuracy_at_three: 0.1860 - val_los
s: 4.9559
Epoch 23/50
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.5924 - accuracy_at_
five: 0.9670 - accuracy_at_three: 0.8852 - loss: 1.1411 - val_accuracy: 0.05
```

```
43 - val_accuracy_at_five: 0.3959 - val_accuracy_at_three: 0.1804 - val_los
s: 4.9255
Epoch 24/50
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.5925 - accuracy_at_
five: 0.9666 - accuracy_at_three: 0.8847 - loss: 1.1418 - val_accuracy: 0.05
64 - val_accuracy_at_five: 0.3802 - val_accuracy_at_three: 0.1717 - val_los
s: 4.9229
Epoch 25/50
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.5916 - accuracy_at_
five: 0.9653 - accuracy_at_three: 0.8836 - loss: 1.1323 - val_accuracy: 0.05
38 - val_accuracy_at_five: 0.3704 - val_accuracy_at_three: 0.1572 - val_los
s: 4.9353
Epoch 26/50
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - accuracy: 0.5899 - accuracy_at
_five: 0.9651 - accuracy_at_three: 0.8807 - loss: 1.1502 - val_accuracy: 0.0
579 - val_accuracy_at_five: 0.3942 - val_accuracy_at_three: 0.1908 - val_los
s: 5.0791
Epoch 27/50
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.5908 - accuracy_at_
five: 0.9646 - accuracy_at_three: 0.8814 - loss: 1.1489 - val_accuracy: 0.04
92 - val_accuracy_at_five: 0.3865 - val_accuracy_at_three: 0.1745 - val_los
s: 5.1233
Epoch 28/50
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 11s 5ms/step - accuracy: 0.5887 - accuracy_at
_five: 0.9656 - accuracy_at_three: 0.8826 - loss: 1.1448 - val_accuracy: 0.0
570 - val_accuracy_at_five: 0.3945 - val_accuracy_at_three: 0.1936 - val_los
s: 5.0317
Epoch 29/50
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 11s 6ms/step - accuracy: 0.5887 - accuracy_at
_five: 0.9651 - accuracy_at_three: 0.8817 - loss: 1.1471 - val_accuracy: 0.0
543 - val_accuracy_at_five: 0.3832 - val_accuracy_at_three: 0.1773 - val_los
s: 5.0598
Epoch 30/50
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 18s 5ms/step - accuracy: 0.5913 - accuracy_at
_five: 0.9656 - accuracy_at_three: 0.8840 - loss: 1.1350 - val_accuracy: 0.0
521 - val_accuracy_at_five: 0.3802 - val_accuracy_at_three: 0.1828 - val_los
s: 5.0946
Epoch 31/50
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.5880 - accuracy_at_
five: 0.9635 - accuracy_at_three: 0.8810 - loss: 1.1451 - val_accuracy: 0.04
96 - val_accuracy_at_five: 0.3458 - val_accuracy_at_three: 0.1587 - val_los
s: 4.9730
Epoch 32/50
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.5864 - accuracy_at_
five: 0.9634 - accuracy_at_three: 0.8782 - loss: 1.1605 - val_accuracy: 0.05
42 - val_accuracy_at_five: 0.3770 - val_accuracy_at_three: 0.1880 - val_los
s: 5.1546
Epoch 33/50
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.5860 - accuracy_at_
five: 0.9629 - accuracy_at_three: 0.8809 - loss: 1.1541 - val_accuracy: 0.05
27 - val_accuracy_at_five: 0.3492 - val_accuracy_at_three: 0.1711 - val_los
s: 5.0703
Epoch 34/50
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - accuracy: 0.5842 - accuracy_at
_five: 0.9611 - accuracy_at_three: 0.8762 - loss: 1.1665 - val_accuracy: 0.0
544 - val_accuracy_at_five: 0.3735 - val_accuracy_at_three: 0.1697 - val_los
```

```
s: 5.2569
Epoch 35/50
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.5843 - accuracy_at_
five: 0.9607 - accuracy_at_three: 0.8754 - loss: 1.1716 - val_accuracy: 0.05
30 - val_accuracy_at_five: 0.3607 - val_accuracy_at_three: 0.1621 - val_los
s: 5.3080
Epoch 36/50
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.5900 - accuracy_at_
five: 0.9620 - accuracy_at_three: 0.8792 - loss: 1.1603 - val_accuracy: 0.05
15 - val_accuracy_at_five: 0.3678 - val_accuracy_at_three: 0.1684 - val_los
s: 5.1910
Epoch 37/50
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - accuracy: 0.5860 - accuracy_at
_five: 0.9627 - accuracy_at_three: 0.8763 - loss: 1.1626 - val_accuracy: 0.0
492 - val_accuracy_at_five: 0.3743 - val_accuracy_at_three: 0.1650 - val_los
s: 5.2388
Epoch 38/50
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 11s 6ms/step - accuracy: 0.5849 - accuracy_at
_five: 0.9628 - accuracy_at_three: 0.8789 - loss: 1.1524 - val_accuracy: 0.0
478 - val_accuracy_at_five: 0.3381 - val_accuracy_at_three: 0.1494 - val_los
s: 5.1192
Epoch 39/50
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.5852 - accuracy_at_
five: 0.9619 - accuracy_at_three: 0.8759 - loss: 1.1622 - val_accuracy: 0.04
92 - val_accuracy_at_five: 0.3655 - val_accuracy_at_three: 0.1630 - val_los
s: 5.1525
Epoch 40/50
1872/1875 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.5872 - accuracy_at_
five: 0.9628 - accuracy_at_three: 0.8782 - loss: 1.1486
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[155], line 1
----> 1 train_model(
      2   model=compile_model(build_function=create_baseline),
      3   name='baseline_resampled',
      4   training_features=[train_moves_resampled, train_features_resample
d],
      5   training_labels=train_labels_resampled,
      6   epochs=50,
      7 )

Cell In[73], line 28, in train_model(model, training_features, training_labe
ls, epochs, callbacks, early_stopping, validation_split, batch_size, class_w
eight, name, log_dir)
     20 date = datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
     21 callbacks.extend([
     22   keras.callbacks.ModelCheckpoint(f'models/{name}{{epoch:02d}}-{{val
_loss:.2f}}.keras'),
     23   keras.callbacks.ModelCheckpoint(f'models/{name}-best.keras', save_
best_only=True, monitor='val_loss'),
     24   keras.callbacks.BackupAndRestore(backup_dir=f'/tmp/backup/{name}--
{date}'),
     25   keras.callbacks.TensorBoard(log_dir=f'{log_dir}/{name}--{date}', h
istogram_freq=1)
     26 ])
---> 28 model.fit(
     29   training_features,
     30   training_labels,
     31   epochs=epochs,
     32   callbacks=callbacks,
     33   validation_split=validation_split,
     34   batch_size=batch_size,
     35   class_weight=class_weight,
     36 )
     37 return model

File ~/.local/lib/python3.10/site-packages/keras/src/utils/traceback_utils.p
y:117, in filter_traceback.<locals>.error_handler(*args, **kwargs)
    115 filtered_tb = None
    116 try:
--> 117     return fn(*args, **kwargs)
    118 except Exception as e:
    119     filtered_tb = _process_traceback_frames(e.__traceback__)

File ~/.local/lib/python3.10/site-packages/keras/src/backend/tensorflow/trai
ner.py:343, in TensorFlowTrainer.fit(self, x, y, batch_size, epochs, verbos
e, callbacks, validation_split, validation_data, shuffle, class_weight, samp
le_weight, initial_epoch, steps_per_epoch, validation_steps, validation_batc
h_size, validation_freq)
    332 if getattr(self, "_eval_epoch_iterator", None) is None:
    333     self._eval_epoch_iterator = TFEpochIterator(
    334         x=val_x,
    335         y=val_y,
   (...)
    341         shuffle=False,
```

```
    342     )
--> 343 val_logs = self.evaluate(
    344     x=val_x,
    345     y=val_y,
    346     sample_weight=val_sample_weight,
    347     batch_size=validation_batch_size or batch_size,
    348     steps=validation_steps,
    349     callbacks=callbacks,
    350     return_dict=True,
    351     _use_cached_eval_dataset=True,
    352 )
    353 val_logs = {
    354     "val_" + name: val for name, val in val_logs.items()
    355 }
    356 epoch_logs.update(val_logs)

File ~/.local/lib/python3.10/site-packages/keras/src/utils/traceback_utils.p
y:117, in filter_traceback.<locals>.error_handler(*args, **kwargs)
    115 filtered_tb = None
    116 try:
--> 117     return fn(*args, **kwargs)
    118 except Exception as e:
    119     filtered_tb = _process_traceback_frames(e.__traceback__)

File ~/.local/lib/python3.10/site-packages/keras/src/backend/tensorflow/trai
ner.py:429, in TensorFlowTrainer.evaluate(self, x, y, batch_size, verbose, s
ample_weight, steps, callbacks, return_dict, **kwargs)
    427 for step, iterator in epoch_iterator.enumerate_epoch():
    428     callbacks.on_test_batch_begin(step)
--> 429     logs = self.test_function(iterator)
    430     logs = self._pythonify_logs(logs)
    431     callbacks.on_test_batch_end(step, logs)

File ~/.local/lib/python3.10/site-packages/tensorflow/python/util/traceback_
utils.py:150, in filter_traceback.<locals>.error_handler(*args, **kwargs)
    148 filtered_tb = None
    149 try:
--> 150   return fn(*args, **kwargs)
    151 except Exception as e:
    152   filtered_tb = _process_traceback_frames(e.__traceback__)

File ~/.local/lib/python3.10/site-packages/tensorflow/python/eager/polymorph
ic_function/polymorphic_function.py:833, in Function.__call__(self, *args, *
*kwds)
    830 compiler = "xla" if self._jit_compile else "nonXla"
    832 with OptionalXlaContext(self._jit_compile):
--> 833   result = self._call(*args, **kwds)
    835 new_tracing_count = self.experimental_get_tracing_count()
    836 without_tracing = (tracing_count == new_tracing_count)

File ~/.local/lib/python3.10/site-packages/tensorflow/python/eager/polymorph
ic_function/polymorphic_function.py:878, in Function._call(self, *args, **kw
ds)
    875 self._lock.release()
    876 # In this case we have not created variables on the first call. So w
e can
```

```
    877 # run the first trace but we should fail if variables are created.
--> 878 results = tracing_compilation.call_function(
    879     args, kwds, self._variable_creation_config
    880 )
    881 if self._created_variables:
    882   raise ValueError("Creating variables on a non-first call to a func
tion"
    883                    " decorated with tf.function.")

File ~/.local/lib/python3.10/site-packages/tensorflow/python/eager/polymorph
ic_function/tracing_compilation.py:139, in call_function(args, kwargs, traci
ng_options)
    137 bound_args = function.function_type.bind(*args, **kwargs)
    138 flat_inputs = function.function_type.unpack_inputs(bound_args)
--> 139 return function._call_flat(  # pylint: disable=protected-access
    140     flat_inputs, captured_inputs=function.captured_inputs
    141 )

File ~/.local/lib/python3.10/site-packages/tensorflow/python/eager/polymorph
ic_function/concrete_function.py:1322, in ConcreteFunction._call_flat(self,
tensor_inputs, captured_inputs)
   1318 possible_gradient_type = gradients_util.PossibleTapeGradientTypes(ar
gs)
   1319 if (possible_gradient_type == gradients_util.POSSIBLE_GRADIENT_TYPES
_NONE
   1320     and executing_eagerly):
   1321   # No tape is watching; skip to running the function.
-> 1322   return self._inference_function.call_preflattened(args)
   1323 forward_backward = self._select_forward_and_backward_functions(
   1324     args,
   1325     possible_gradient_type,
   1326     executing_eagerly)
   1327 forward_function, args_with_tangents = forward_backward.forward()

File ~/.local/lib/python3.10/site-packages/tensorflow/python/eager/polymorph
ic_function/atomic_function.py:216, in AtomicFunction.call_preflattened(sel
f, args)
    214 def call_preflattened(self, args: Sequence[core.Tensor]) -> Any:
    215   """Calls with flattened tensor inputs and returns the structured o
utput."""
--> 216   flat_outputs = self.call_flat(*args)
    217   return self.function_type.pack_output(flat_outputs)

File ~/.local/lib/python3.10/site-packages/tensorflow/python/eager/polymorph
ic_function/atomic_function.py:251, in AtomicFunction.call_flat(self, *args)
    249 with record.stop_recording():
    250   if self._bound_context.executing_eagerly():
--> 251     outputs = self._bound_context.call_function(
    252         self.name,
    253         list(args),
    254         len(self.function_type.flat_outputs),
    255     )
    256   else:
    257     outputs = make_call_op_in_graph(
    258         self,
    259         list(args),
```

```
    260         self._bound_context.function_call_options.as_attrs(),
    261     )

File ~/.local/lib/python3.10/site-packages/tensorflow/python/eager/context.p
y:1552, in Context.call_function(self, name, tensor_inputs, num_outputs)
   1550 cancellation_context = cancellation.context()
   1551 if cancellation_context is None:
-> 1552   outputs = execute.execute(
   1553       name.decode("utf-8"),
   1554       num_outputs=num_outputs,
   1555       inputs=tensor_inputs,
   1556       attrs=attrs,
   1557       ctx=self,
   1558   )
   1559 else:
   1560   outputs = execute.execute_with_cancellation(
   1561       name.decode("utf-8"),
   1562       num_outputs=num_outputs,
   (...)
   1566       cancellation_manager=cancellation_context,
   1567   )

File ~/.local/lib/python3.10/site-packages/tensorflow/python/eager/execute.p
y:53, in quick_execute(op_name, num_outputs, inputs, attrs, ctx, name)
     51 try:
     52   ctx.ensure_initialized()
---> 53   tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name, op_n
ame,
     54                                       inputs, attrs, num_outputs)
     55 except core._NotOkStatusException as e:
     56   if name is not None:

KeyboardInterrupt:
```

The training accuracy is the best of all, but the validation accuracy is the worst of all... training and validation lost curves keep the same distance for every epoch but with an important gap

Treating routes as images: Convolution

During Data analysis, we chose to plot our routes as images, with the three separate channels representing the type of hold (start, middle, end). So, if our routes can be interpreted as images, why can't our problem be an image classification problem? That's what we will explore using Convolutional Neural Networks (CNN).

Testing multiple configurations

Here, we try several different configurations using many techniques :

- Conv2D vs SeparableConv2D layers
- Number and shape of layers
- Padding, strides
- Residual connections
- Normalization in the middle of the model

In [150…

```
def create_convolutional():
  shape = [32, 64, 128, 256]

  moves_inputs = keras.Input(shape=MOVES_SHAPE, name="moves")
  features_inputs = keras.Input(shape=(nb_features,), name="features")

  x = keras.layers.Dense(64, activation='relu')(features_inputs)
  features_outputs = keras.layers.Dense(64, activation='relu')(x)

  # The assumption for using depth wise-separable convolution is channel inc
  # The three channels are part of the same route, and only indicate the sta
  y = keras.layers.Conv2D(filters=32, kernel_size=3, padding='same', use_bia

  for i, filters in enumerate(shape):
    connection = y

    y = tf.keras.layers.BatchNormalization()(y)
    y = tf.keras.layers.Activation('relu')(y)
    y = keras.layers.SeparableConv2D(filters=filters, kernel_size=3, padding

    y = keras.layers.MaxPool2D(pool_size=2, padding='same')(y)

    # Residual connection fit
    connection = keras.layers.Conv2D(filters=filters, kernel_size=1, strides

    y = keras.layers.add((y, connection))

  moves_outputs = keras.layers.GlobalAveragePooling2D()(y)

  x = keras.layers.concatenate([features_outputs, moves_outputs])

  x = keras.layers.Dense(64, activation='relu')(x)
  x = keras.layers.Dropout(0.5)(x)
  outputs = keras.layers.Dense(nb_labels, activation='softmax')(x)

  return keras.Model(inputs=[moves_inputs, features_inputs], outputs=outputs
```

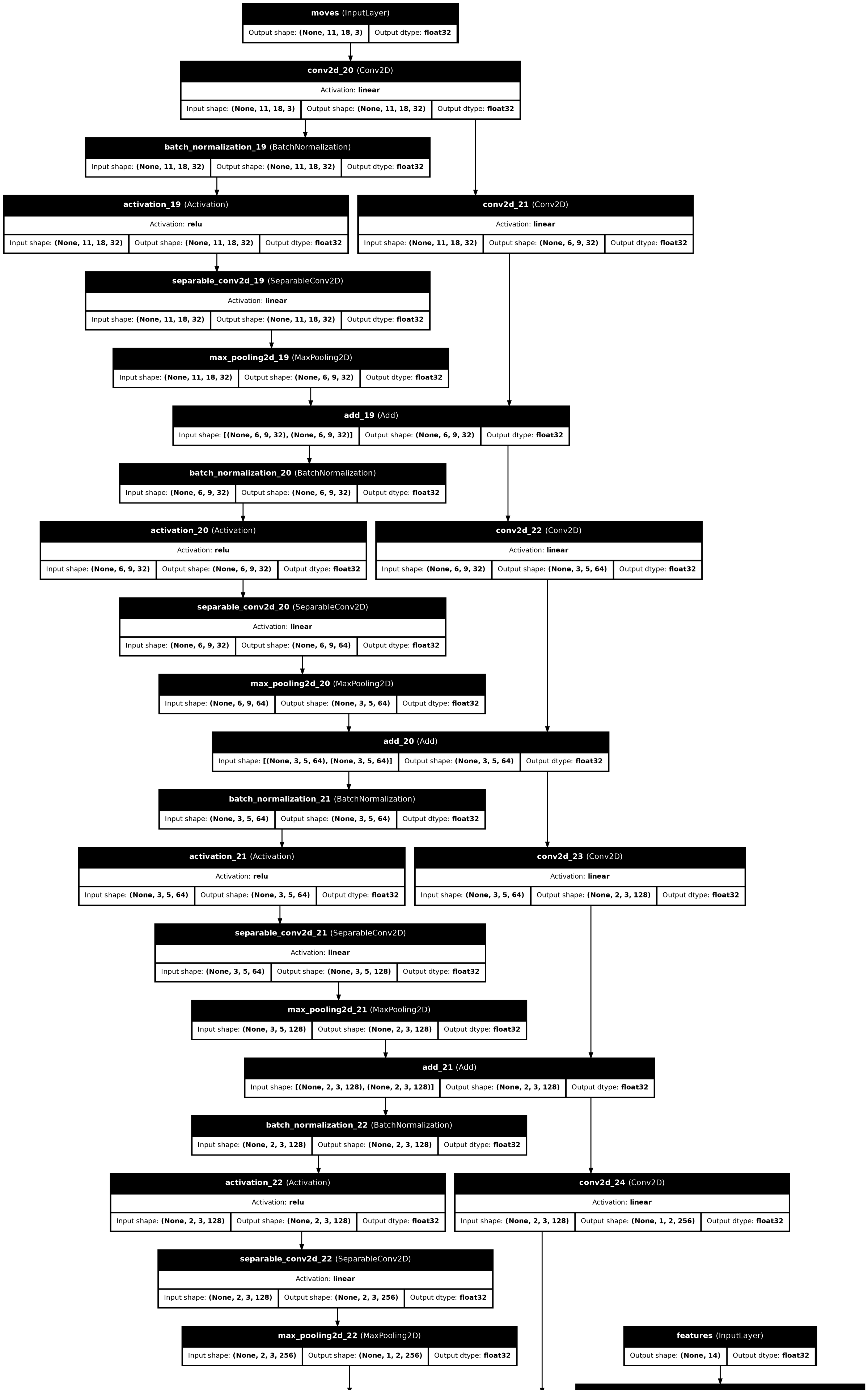
In [151…

```
convolution_model = compile_model(build_function=create_convolutional)
```

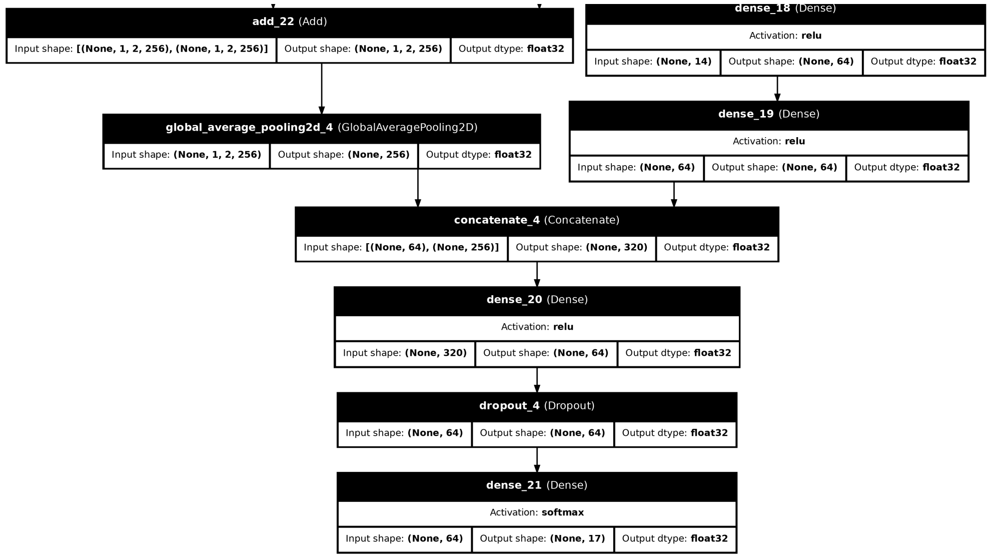
In [152…

```
plot_model(convolution_model)
Image('model.png')
```

Out[152…



moves (InputLayer)
Output shape: (None, 11, 18, 3) | Output dtype: float32
conv2d_20 (Conv2D)
Activation: linear
Input shape: (None, 11, 18, 3) | Output shape: (None, 11, 18, 32) | Output dtype: float32
batch_normalization_19 (BatchNormalization)
Input shape: (None, 11, 18, 32) | Output shape: (None, 11, 18, 32) | Output dtype: float32
activation_19 (Activation)
Activation: relu
Input shape: (None, 11, 18, 32) | Output shape: (None, 11, 18, 32) | Output dtype: float32
conv2d_21 (Conv2D)
Activation: linear
Input shape: (None, 11, 18, 32) | Output shape: (None, 6, 9, 32) | Output dtype: float32
separable_conv2d_19 (SeparableConv2D)
Activation: linear
Input shape: (None, 11, 18, 32) | Output shape: (None, 11, 18, 32) | Output dtype: float32
max_pooling2d_19 (MaxPooling2D)
Input shape: (None, 11, 18, 32) | Output shape: (None, 6, 9, 32) | Output dtype: float32
add_19 (Add)
Input shape: [(None, 6, 9, 32), (None, 6, 9, 32)] | Output shape: (None, 6, 9, 32) | Output dtype: float32
batch_normalization_20 (BatchNormalization)
Input shape: (None, 6, 9, 32) | Output shape: (None, 6, 9, 32) | Output dtype: float32
activation_20 (Activation)
Activation: relu
Input shape: (None, 6, 9, 32) | Output shape: (None, 6, 9, 32) | Output dtype: float32
conv2d_22 (Conv2D)
Activation: linear
Input shape: (None, 6, 9, 32) | Output shape: (None, 3, 5, 64) | Output dtype: float32
separable_conv2d_20 (SeparableConv2D)
Activation: linear
Input shape: (None, 6, 9, 32) | Output shape: (None, 6, 9, 64) | Output dtype: float32
max_pooling2d_20 (MaxPooling2D)
Input shape: (None, 6, 9, 64) | Output shape: (None, 3, 5, 64) | Output dtype: float32
add_20 (Add)
Input shape: [(None, 3, 5, 64), (None, 3, 5, 64)] | Output shape: (None, 3, 5, 64) | Output dtype: float32
batch_normalization_21 (BatchNormalization)
Input shape: (None, 3, 5, 64) | Output shape: (None, 3, 5, 64) | Output dtype: float32
activation_21 (Activation)
Activation: relu
Input shape: (None, 3, 5, 64) | Output shape: (None, 3, 5, 64) | Output dtype: float32
conv2d_23 (Conv2D)
Activation: linear
Input shape: (None, 3, 5, 64) | Output shape: (None, 2, 3, 128) | Output dtype: float32
separable_conv2d_21 (SeparableConv2D)
Activation: linear
Input shape: (None, 3, 5, 64) | Output shape: (None, 3, 5, 128) | Output dtype: float32
max_pooling2d_21 (MaxPooling2D)
Input shape: (None, 3, 5, 128) | Output shape: (None, 2, 3, 128) | Output dtype: float32
add_21 (Add)
Input shape: [(None, 2, 3, 128), (None, 2, 3, 128)] | Output shape: (None, 2, 3, 128) | Output dtype: float32
batch_normalization_22 (BatchNormalization)
Input shape: (None, 2, 3, 128) | Output shape: (None, 2, 3, 128) | Output dtype: float32
activation_22 (Activation)
Activation: relu
Input shape: (None, 2, 3, 128) | Output shape: (None, 2, 3, 128) | Output dtype: float32
conv2d_24 (Conv2D)
Activation: linear
Input shape: (None, 2, 3, 128) | Output shape: (None, 1, 2, 256) | Output dtype: float32
separable_conv2d_22 (SeparableConv2D)
Activation: linear
Input shape: (None, 2, 3, 128) | Output shape: (None, 2, 3, 256) | Output dtype: float32
max_pooling2d_22 (MaxPooling2D)
Input shape: (None, 2, 3, 256) | Output shape: (None, 1, 2, 256) | Output dtype: float32
features (InputLayer)
Output shape: (None, 14) | Output dtype: float32



In [44]:
```
train_model(
  model=convolution_model,
  name='convolution',
  training_features=[train_moves, train_features],
  training_labels=train_labels,
  epochs=70
)
```

```
Epoch 1/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 28s 13ms/step - accuracy: 0.2412 - accuracy_a
t_five: 0.7006 - accuracy_at_three: 0.5141 - loss: 2.2665 - val_accuracy: 0.
3433 - val_accuracy_at_five: 0.8550 - val_accuracy_at_three: 0.6759 - val_lo
ss: 1.7593
Epoch 2/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 35s 13ms/step - accuracy: 0.3249 - accuracy_a
t_five: 0.8336 - accuracy_at_three: 0.6500 - loss: 1.8466 - val_accuracy: 0.
3425 - val_accuracy_at_five: 0.8564 - val_accuracy_at_three: 0.6765 - val_lo
ss: 1.7617
Epoch 3/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 69s 49ms/step - accuracy: 0.3358 - accuracy_a
t_five: 0.8497 - accuracy_at_three: 0.6685 - loss: 1.7860 - val_accuracy: 0.
3418 - val_accuracy_at_five: 0.8496 - val_accuracy_at_three: 0.6772 - val_lo
ss: 1.7754
Epoch 4/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 34s 15ms/step - accuracy: 0.3516 - accuracy_a
t_five: 0.8643 - accuracy_at_three: 0.6888 - loss: 1.7468 - val_accuracy: 0.
3692 - val_accuracy_at_five: 0.8820 - val_accuracy_at_three: 0.7134 - val_lo
ss: 1.6759
Epoch 5/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 22s 15ms/step - accuracy: 0.3536 - accuracy_a
t_five: 0.8713 - accuracy_at_three: 0.6941 - loss: 1.7242 - val_accuracy: 0.
3704 - val_accuracy_at_five: 0.8863 - val_accuracy_at_three: 0.7168 - val_lo
ss: 1.6718
Epoch 6/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 26s 18ms/step - accuracy: 0.3558 - accuracy_a
t_five: 0.8724 - accuracy_at_three: 0.6936 - loss: 1.7211 - val_accuracy: 0.
3657 - val_accuracy_at_five: 0.8787 - val_accuracy_at_three: 0.7077 - val_lo
ss: 1.6797
Epoch 7/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 49s 34ms/step - accuracy: 0.3567 - accuracy_a
t_five: 0.8763 - accuracy_at_three: 0.6990 - loss: 1.7145 - val_accuracy: 0.
3672 - val_accuracy_at_five: 0.8799 - val_accuracy_at_three: 0.7082 - val_lo
ss: 1.6764
Epoch 8/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 61s 20ms/step - accuracy: 0.3586 - accuracy_a
t_five: 0.8786 - accuracy_at_three: 0.7033 - loss: 1.7029 - val_accuracy: 0.
3697 - val_accuracy_at_five: 0.8913 - val_accuracy_at_three: 0.7224 - val_lo
ss: 1.6595
Epoch 9/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 34s 15ms/step - accuracy: 0.3607 - accuracy_a
t_five: 0.8817 - accuracy_at_three: 0.7070 - loss: 1.6938 - val_accuracy: 0.
3769 - val_accuracy_at_five: 0.8923 - val_accuracy_at_three: 0.7231 - val_lo
ss: 1.6552
Epoch 10/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 32s 22ms/step - accuracy: 0.3636 - accuracy_a
t_five: 0.8839 - accuracy_at_three: 0.7087 - loss: 1.6870 - val_accuracy: 0.
3751 - val_accuracy_at_five: 0.8927 - val_accuracy_at_three: 0.7227 - val_lo
ss: 1.6502
Epoch 11/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 49s 34ms/step - accuracy: 0.3642 - accuracy_a
t_five: 0.8826 - accuracy_at_three: 0.7120 - loss: 1.6804 - val_accuracy: 0.
3364 - val_accuracy_at_five: 0.8378 - val_accuracy_at_three: 0.6621 - val_lo
ss: 1.8123
Epoch 12/70
```

```
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 21s 14ms/step - accuracy: 0.3646 - accuracy_a
t_five: 0.8830 - accuracy_at_three: 0.7102 - loss: 1.6822 - val_accuracy: 0.
3749 - val_accuracy_at_five: 0.8933 - val_accuracy_at_three: 0.7251 - val_lo
ss: 1.6442
Epoch 13/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 18s 13ms/step - accuracy: 0.3625 - accuracy_a
t_five: 0.8854 - accuracy_at_three: 0.7132 - loss: 1.6788 - val_accuracy: 0.
3629 - val_accuracy_at_five: 0.8795 - val_accuracy_at_three: 0.7048 - val_lo
ss: 1.7075
Epoch 14/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 18s 13ms/step - accuracy: 0.3650 - accuracy_a
t_five: 0.8844 - accuracy_at_three: 0.7146 - loss: 1.6695 - val_accuracy: 0.
3775 - val_accuracy_at_five: 0.8956 - val_accuracy_at_three: 0.7260 - val_lo
ss: 1.6333
Epoch 15/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 16s 11ms/step - accuracy: 0.3686 - accuracy_a
t_five: 0.8861 - accuracy_at_three: 0.7143 - loss: 1.6700 - val_accuracy: 0.
3784 - val_accuracy_at_five: 0.8955 - val_accuracy_at_three: 0.7274 - val_lo
ss: 1.6363
Epoch 16/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 16s 11ms/step - accuracy: 0.3711 - accuracy_a
t_five: 0.8892 - accuracy_at_three: 0.7176 - loss: 1.6568 - val_accuracy: 0.
3678 - val_accuracy_at_five: 0.8867 - val_accuracy_at_three: 0.7161 - val_lo
ss: 1.6719
Epoch 17/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 18s 13ms/step - accuracy: 0.3663 - accuracy_a
t_five: 0.8881 - accuracy_at_three: 0.7150 - loss: 1.6627 - val_accuracy: 0.
3664 - val_accuracy_at_five: 0.8854 - val_accuracy_at_three: 0.7134 - val_lo
ss: 1.6637
Epoch 18/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 17s 12ms/step - accuracy: 0.3674 - accuracy_a
t_five: 0.8882 - accuracy_at_three: 0.7155 - loss: 1.6594 - val_accuracy: 0.
3819 - val_accuracy_at_five: 0.8990 - val_accuracy_at_three: 0.7305 - val_lo
ss: 1.6351
Epoch 19/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 19s 13ms/step - accuracy: 0.3696 - accuracy_a
t_five: 0.8905 - accuracy_at_three: 0.7198 - loss: 1.6576 - val_accuracy: 0.
3739 - val_accuracy_at_five: 0.8962 - val_accuracy_at_three: 0.7282 - val_lo
ss: 1.6432
Epoch 20/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 27s 19ms/step - accuracy: 0.3673 - accuracy_a
t_five: 0.8875 - accuracy_at_three: 0.7165 - loss: 1.6654 - val_accuracy: 0.
3734 - val_accuracy_at_five: 0.8808 - val_accuracy_at_three: 0.7151 - val_lo
ss: 1.6859
Epoch 21/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 18s 13ms/step - accuracy: 0.3699 - accuracy_a
t_five: 0.8907 - accuracy_at_three: 0.7213 - loss: 1.6494 - val_accuracy: 0.
3793 - val_accuracy_at_five: 0.8965 - val_accuracy_at_three: 0.7274 - val_lo
ss: 1.6441
Epoch 22/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 16s 11ms/step - accuracy: 0.3698 - accuracy_a
t_five: 0.8895 - accuracy_at_three: 0.7184 - loss: 1.6518 - val_accuracy: 0.
3644 - val_accuracy_at_five: 0.8828 - val_accuracy_at_three: 0.7116 - val_lo
ss: 1.6766
Epoch 23/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 16s 11ms/step - accuracy: 0.3659 - accuracy_a
```

```
t_five: 0.8891 - accuracy_at_three: 0.7186 - loss: 1.6643 - val_accuracy: 0.
3764 - val_accuracy_at_five: 0.8983 - val_accuracy_at_three: 0.7320 - val_lo
ss: 1.6301
Epoch 24/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 23s 13ms/step - accuracy: 0.3653 - accuracy_a
t_five: 0.8910 - accuracy_at_three: 0.7190 - loss: 1.6572 - val_accuracy: 0.
3767 - val_accuracy_at_five: 0.8937 - val_accuracy_at_three: 0.7294 - val_lo
ss: 1.6420
Epoch 25/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 32s 21ms/step - accuracy: 0.3704 - accuracy_a
t_five: 0.8885 - accuracy_at_three: 0.7203 - loss: 1.6553 - val_accuracy: 0.
3767 - val_accuracy_at_five: 0.8933 - val_accuracy_at_three: 0.7243 - val_lo
ss: 1.6477
Epoch 26/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 34s 16ms/step - accuracy: 0.3706 - accuracy_a
t_five: 0.8929 - accuracy_at_three: 0.7217 - loss: 1.6508 - val_accuracy: 0.
3602 - val_accuracy_at_five: 0.8758 - val_accuracy_at_three: 0.6975 - val_lo
ss: 1.7080
Epoch 27/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 24s 17ms/step - accuracy: 0.3720 - accuracy_a
t_five: 0.8905 - accuracy_at_three: 0.7223 - loss: 1.6461 - val_accuracy: 0.
3771 - val_accuracy_at_five: 0.9014 - val_accuracy_at_three: 0.7317 - val_lo
ss: 1.6515
Epoch 28/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 35s 25ms/step - accuracy: 0.3714 - accuracy_a
t_five: 0.8926 - accuracy_at_three: 0.7217 - loss: 1.6496 - val_accuracy: 0.
3750 - val_accuracy_at_five: 0.8974 - val_accuracy_at_three: 0.7300 - val_lo
ss: 1.6377
Epoch 29/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 20s 14ms/step - accuracy: 0.3707 - accuracy_a
t_five: 0.8919 - accuracy_at_three: 0.7218 - loss: 1.6485 - val_accuracy: 0.
3816 - val_accuracy_at_five: 0.8971 - val_accuracy_at_three: 0.7347 - val_lo
ss: 1.6400
Epoch 30/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 17s 12ms/step - accuracy: 0.3718 - accuracy_a
t_five: 0.8915 - accuracy_at_three: 0.7202 - loss: 1.6505 - val_accuracy: 0.
3786 - val_accuracy_at_five: 0.8953 - val_accuracy_at_three: 0.7289 - val_lo
ss: 1.6396
Epoch 31/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 19s 13ms/step - accuracy: 0.3674 - accuracy_a
t_five: 0.8919 - accuracy_at_three: 0.7215 - loss: 1.6500 - val_accuracy: 0.
3773 - val_accuracy_at_five: 0.8993 - val_accuracy_at_three: 0.7318 - val_lo
ss: 1.6549
Epoch 32/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 18s 13ms/step - accuracy: 0.3676 - accuracy_a
t_five: 0.8909 - accuracy_at_three: 0.7207 - loss: 1.6518 - val_accuracy: 0.
3722 - val_accuracy_at_five: 0.8936 - val_accuracy_at_three: 0.7237 - val_lo
ss: 1.7006
Epoch 33/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 20s 14ms/step - accuracy: 0.3706 - accuracy_a
t_five: 0.8902 - accuracy_at_three: 0.7210 - loss: 1.6532 - val_accuracy: 0.
3778 - val_accuracy_at_five: 0.8961 - val_accuracy_at_three: 0.7300 - val_lo
ss: 1.6503
Epoch 34/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 24s 16ms/step - accuracy: 0.3716 - accuracy_a
t_five: 0.8968 - accuracy_at_three: 0.7253 - loss: 1.6421 - val_accuracy: 0.
```

```
3789 - val_accuracy_at_five: 0.8974 - val_accuracy_at_three: 0.7265 - val_lo
ss: 1.7043
Epoch 35/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 35s 24ms/step - accuracy: 0.3738 - accuracy_a
t_five: 0.8929 - accuracy_at_three: 0.7225 - loss: 1.6446 - val_accuracy: 0.
3788 - val_accuracy_at_five: 0.8945 - val_accuracy_at_three: 0.7278 - val_lo
ss: 1.6591
Epoch 36/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 18s 13ms/step - accuracy: 0.3673 - accuracy_a
t_five: 0.8941 - accuracy_at_three: 0.7237 - loss: 1.6495 - val_accuracy: 0.
3753 - val_accuracy_at_five: 0.8951 - val_accuracy_at_three: 0.7281 - val_lo
ss: 1.6495
Epoch 37/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 41s 28ms/step - accuracy: 0.3711 - accuracy_a
t_five: 0.8926 - accuracy_at_three: 0.7213 - loss: 1.6451 - val_accuracy: 0.
3724 - val_accuracy_at_five: 0.8951 - val_accuracy_at_three: 0.7258 - val_lo
ss: 1.6530
Epoch 38/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 17s 12ms/step - accuracy: 0.3744 - accuracy_a
t_five: 0.8936 - accuracy_at_three: 0.7240 - loss: 1.6410 - val_accuracy: 0.
3745 - val_accuracy_at_five: 0.8902 - val_accuracy_at_three: 0.7244 - val_lo
ss: 1.6572
Epoch 39/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 20s 14ms/step - accuracy: 0.3733 - accuracy_a
t_five: 0.8935 - accuracy_at_three: 0.7235 - loss: 1.6415 - val_accuracy: 0.
3719 - val_accuracy_at_five: 0.8916 - val_accuracy_at_three: 0.7244 - val_lo
ss: 1.6703
Epoch 40/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 16s 11ms/step - accuracy: 0.3731 - accuracy_a
t_five: 0.8935 - accuracy_at_three: 0.7274 - loss: 1.6401 - val_accuracy: 0.
3781 - val_accuracy_at_five: 0.8966 - val_accuracy_at_three: 0.7276 - val_lo
ss: 1.6530
Epoch 41/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 19s 13ms/step - accuracy: 0.3728 - accuracy_a
t_five: 0.8927 - accuracy_at_three: 0.7240 - loss: 1.6472 - val_accuracy: 0.
3684 - val_accuracy_at_five: 0.8795 - val_accuracy_at_three: 0.7134 - val_lo
ss: 1.6848
Epoch 42/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 17s 12ms/step - accuracy: 0.3723 - accuracy_a
t_five: 0.8952 - accuracy_at_three: 0.7242 - loss: 1.6446 - val_accuracy: 0.
3761 - val_accuracy_at_five: 0.8880 - val_accuracy_at_three: 0.7235 - val_lo
ss: 1.6825
Epoch 43/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 18s 13ms/step - accuracy: 0.3745 - accuracy_a
t_five: 0.8956 - accuracy_at_three: 0.7261 - loss: 1.6407 - val_accuracy: 0.
3576 - val_accuracy_at_five: 0.8806 - val_accuracy_at_three: 0.7046 - val_lo
ss: 1.6986
Epoch 44/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 21s 14ms/step - accuracy: 0.3742 - accuracy_a
t_five: 0.8930 - accuracy_at_three: 0.7230 - loss: 1.6408 - val_accuracy: 0.
3816 - val_accuracy_at_five: 0.8899 - val_accuracy_at_three: 0.7266 - val_lo
ss: 1.6895
Epoch 45/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 15s 11ms/step - accuracy: 0.3721 - accuracy_a
t_five: 0.8942 - accuracy_at_three: 0.7204 - loss: 1.6514 - val_accuracy: 0.
3731 - val_accuracy_at_five: 0.8974 - val_accuracy_at_three: 0.7300 - val_lo
```

```
ss: 1.6506
Epoch 46/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 14s 10ms/step - accuracy: 0.3708 - accuracy_a
t_five: 0.8897 - accuracy_at_three: 0.7204 - loss: 1.6573 - val_accuracy: 0.
3679 - val_accuracy_at_five: 0.8897 - val_accuracy_at_three: 0.7186 - val_lo
ss: 1.6981
Epoch 47/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 14s 10ms/step - accuracy: 0.3722 - accuracy_a
t_five: 0.8938 - accuracy_at_three: 0.7213 - loss: 1.6511 - val_accuracy: 0.
3774 - val_accuracy_at_five: 0.8956 - val_accuracy_at_three: 0.7321 - val_lo
ss: 1.6671
Epoch 48/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 15s 11ms/step - accuracy: 0.3712 - accuracy_a
t_five: 0.8911 - accuracy_at_three: 0.7186 - loss: 1.6577 - val_accuracy: 0.
3801 - val_accuracy_at_five: 0.8979 - val_accuracy_at_three: 0.7320 - val_lo
ss: 1.6854
Epoch 49/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 19s 13ms/step - accuracy: 0.3678 - accuracy_a
t_five: 0.8922 - accuracy_at_three: 0.7220 - loss: 1.6516 - val_accuracy: 0.
3718 - val_accuracy_at_five: 0.8940 - val_accuracy_at_three: 0.7241 - val_lo
ss: 1.6728
Epoch 50/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 16s 11ms/step - accuracy: 0.3700 - accuracy_a
t_five: 0.8922 - accuracy_at_three: 0.7210 - loss: 1.6562 - val_accuracy: 0.
3807 - val_accuracy_at_five: 0.8986 - val_accuracy_at_three: 0.7317 - val_lo
ss: 1.6525
Epoch 51/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 14s 9ms/step - accuracy: 0.3716 - accuracy_at
_five: 0.8929 - accuracy_at_three: 0.7213 - loss: 1.6503 - val_accuracy: 0.3
779 - val_accuracy_at_five: 0.8981 - val_accuracy_at_three: 0.7303 - val_los
s: 1.6516
Epoch 52/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 15s 10ms/step - accuracy: 0.3737 - accuracy_a
t_five: 0.8921 - accuracy_at_three: 0.7224 - loss: 1.6496 - val_accuracy: 0.
3722 - val_accuracy_at_five: 0.8940 - val_accuracy_at_three: 0.7252 - val_lo
ss: 1.6603
Epoch 53/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 13s 9ms/step - accuracy: 0.3689 - accuracy_at
_five: 0.8929 - accuracy_at_three: 0.7210 - loss: 1.6539 - val_accuracy: 0.3
657 - val_accuracy_at_five: 0.8857 - val_accuracy_at_three: 0.7134 - val_los
s: 1.8124
Epoch 54/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 18s 13ms/step - accuracy: 0.3724 - accuracy_a
t_five: 0.8911 - accuracy_at_three: 0.7181 - loss: 1.6572 - val_accuracy: 0.
3748 - val_accuracy_at_five: 0.8911 - val_accuracy_at_three: 0.7196 - val_lo
ss: 1.6748
Epoch 55/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 16s 11ms/step - accuracy: 0.3717 - accuracy_a
t_five: 0.8908 - accuracy_at_three: 0.7206 - loss: 1.6565 - val_accuracy: 0.
3617 - val_accuracy_at_five: 0.8831 - val_accuracy_at_three: 0.7065 - val_lo
ss: 1.7665
Epoch 56/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 23s 16ms/step - accuracy: 0.3690 - accuracy_a
t_five: 0.8893 - accuracy_at_three: 0.7203 - loss: 1.6639 - val_accuracy: 0.
3798 - val_accuracy_at_five: 0.8952 - val_accuracy_at_three: 0.7282 - val_lo
ss: 1.6841
```

```
Epoch 57/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 22s 15ms/step - accuracy: 0.3690 - accuracy_a
t_five: 0.8901 - accuracy_at_three: 0.7175 - loss: 1.6635 - val_accuracy: 0.
3782 - val_accuracy_at_five: 0.8966 - val_accuracy_at_three: 0.7292 - val_lo
ss: 1.6791
Epoch 58/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 16s 11ms/step - accuracy: 0.3685 - accuracy_a
t_five: 0.8891 - accuracy_at_three: 0.7175 - loss: 1.6694 - val_accuracy: 0.
3740 - val_accuracy_at_five: 0.8955 - val_accuracy_at_three: 0.7309 - val_lo
ss: 1.6688
Epoch 59/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 18s 12ms/step - accuracy: 0.3678 - accuracy_a
t_five: 0.8881 - accuracy_at_three: 0.7147 - loss: 1.6704 - val_accuracy: 0.
3719 - val_accuracy_at_five: 0.8868 - val_accuracy_at_three: 0.7201 - val_lo
ss: 1.7014
Epoch 60/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 16s 11ms/step - accuracy: 0.3649 - accuracy_a
t_five: 0.8887 - accuracy_at_three: 0.7147 - loss: 1.6814 - val_accuracy: 0.
3688 - val_accuracy_at_five: 0.8894 - val_accuracy_at_three: 0.7196 - val_lo
ss: 1.6861
Epoch 61/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 18s 13ms/step - accuracy: 0.3661 - accuracy_a
t_five: 0.8873 - accuracy_at_three: 0.7154 - loss: 1.6718 - val_accuracy: 0.
3726 - val_accuracy_at_five: 0.8907 - val_accuracy_at_three: 0.7254 - val_lo
ss: 1.7232
Epoch 62/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 16s 11ms/step - accuracy: 0.3667 - accuracy_a
t_five: 0.8868 - accuracy_at_three: 0.7138 - loss: 1.6775 - val_accuracy: 0.
3611 - val_accuracy_at_five: 0.8695 - val_accuracy_at_three: 0.6964 - val_lo
ss: 1.7501
Epoch 63/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 16s 11ms/step - accuracy: 0.3678 - accuracy_a
t_five: 0.8891 - accuracy_at_three: 0.7171 - loss: 1.6727 - val_accuracy: 0.
3707 - val_accuracy_at_five: 0.8892 - val_accuracy_at_three: 0.7195 - val_lo
ss: 1.7088
Epoch 64/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 19s 10ms/step - accuracy: 0.3684 - accuracy_a
t_five: 0.8871 - accuracy_at_three: 0.7130 - loss: 1.6799 - val_accuracy: 0.
3748 - val_accuracy_at_five: 0.8917 - val_accuracy_at_three: 0.7238 - val_lo
ss: 1.6704
Epoch 65/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 15s 11ms/step - accuracy: 0.3670 - accuracy_a
t_five: 0.8853 - accuracy_at_three: 0.7113 - loss: 1.6798 - val_accuracy: 0.
3740 - val_accuracy_at_five: 0.8915 - val_accuracy_at_three: 0.7245 - val_lo
ss: 1.6747
Epoch 66/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 16s 11ms/step - accuracy: 0.3638 - accuracy_a
t_five: 0.8847 - accuracy_at_three: 0.7123 - loss: 1.6890 - val_accuracy: 0.
3721 - val_accuracy_at_five: 0.8857 - val_accuracy_at_three: 0.7165 - val_lo
ss: 1.7152
Epoch 67/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 15s 10ms/step - accuracy: 0.3673 - accuracy_a
t_five: 0.8851 - accuracy_at_three: 0.7117 - loss: 1.6849 - val_accuracy: 0.
3661 - val_accuracy_at_five: 0.8812 - val_accuracy_at_three: 0.7048 - val_lo
ss: 1.7068
Epoch 68/70
```

```
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 18s 13ms/step - accuracy: 0.3601 - accuracy_a
t_five: 0.8818 - accuracy_at_three: 0.7071 - loss: 1.7008 - val_accuracy: 0.
3763 - val_accuracy_at_five: 0.8881 - val_accuracy_at_three: 0.7217 - val_lo
ss: 1.6939
Epoch 69/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 21s 15ms/step - accuracy: 0.3622 - accuracy_a
t_five: 0.8844 - accuracy_at_three: 0.7100 - loss: 1.6939 - val_accuracy: 0.
3748 - val_accuracy_at_five: 0.8892 - val_accuracy_at_three: 0.7185 - val_lo
ss: 1.6782
Epoch 70/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 38s 12ms/step - accuracy: 0.3607 - accuracy_a
t_five: 0.8815 - accuracy_at_three: 0.7085 - loss: 1.6946 - val_accuracy: 0.
3476 - val_accuracy_at_five: 0.8650 - val_accuracy_at_three: 0.6834 - val_lo
ss: 1.7899
```

Out[44]:
```
<Functional name=functional_2, built=True>
```

Observations

- Overfitting is far away, many epochs can be achieved
- Padding don't cause underfitting, but augmenting the end dense network yes
- Better performances by normalizing after layers and adding residual connections

Using a learning rate schedule

In [154…
```
convolution_model = compile_model(build_function=create_convolutional, learr
train_model(
  model=convolution_model,
  name='convolution-schedule',
  # class_weight=class_weight_dict,
  callbacks=[
    keras.callbacks.ReduceLROnPlateau(
      monitor="val_loss",
      factor=0.4,
      patience=2,
      min_lr=1e-6
    )
  ],
  training_features=[train_moves, train_features],
  training_labels=train_labels,
  epochs=70
)
```

```
Epoch 1/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 47s 30ms/step - accuracy: 0.2121 - accuracy_a
t_five: 0.6450 - accuracy_at_three: 0.4580 - loss: 2.4094 - val_accuracy: 0.
3224 - val_accuracy_at_five: 0.8201 - val_accuracy_at_three: 0.6398 - val_lo
ss: 1.9163 - learning_rate: 1.0000e-04
Epoch 2/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 52s 36ms/step - accuracy: 0.3039 - accuracy_a
t_five: 0.7971 - accuracy_at_three: 0.5969 - loss: 1.9907 - val_accuracy: 0.
3378 - val_accuracy_at_five: 0.8603 - val_accuracy_at_three: 0.6820 - val_lo
ss: 1.7765 - learning_rate: 1.0000e-04
Epoch 3/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 49s 34ms/step - accuracy: 0.3255 - accuracy_a
t_five: 0.8370 - accuracy_at_three: 0.6443 - loss: 1.8572 - val_accuracy: 0.
3438 - val_accuracy_at_five: 0.8760 - val_accuracy_at_three: 0.6953 - val_lo
ss: 1.7161 - learning_rate: 1.0000e-04
Epoch 4/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 80s 33ms/step - accuracy: 0.3373 - accuracy_a
t_five: 0.8569 - accuracy_at_three: 0.6706 - loss: 1.7885 - val_accuracy: 0.
3632 - val_accuracy_at_five: 0.8839 - val_accuracy_at_three: 0.7126 - val_lo
ss: 1.6779 - learning_rate: 1.0000e-04
Epoch 5/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 49s 34ms/step - accuracy: 0.3450 - accuracy_a
t_five: 0.8688 - accuracy_at_three: 0.6857 - loss: 1.7444 - val_accuracy: 0.
3706 - val_accuracy_at_five: 0.8904 - val_accuracy_at_three: 0.7197 - val_lo
ss: 1.6591 - learning_rate: 1.0000e-04
Epoch 6/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 50s 35ms/step - accuracy: 0.3562 - accuracy_a
t_five: 0.8787 - accuracy_at_three: 0.6976 - loss: 1.7124 - val_accuracy: 0.
3703 - val_accuracy_at_five: 0.8907 - val_accuracy_at_three: 0.7223 - val_lo
ss: 1.6445 - learning_rate: 1.0000e-04
Epoch 7/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 77s 31ms/step - accuracy: 0.3626 - accuracy_a
t_five: 0.8858 - accuracy_at_three: 0.7109 - loss: 1.6874 - val_accuracy: 0.
3736 - val_accuracy_at_five: 0.8938 - val_accuracy_at_three: 0.7273 - val_lo
ss: 1.6333 - learning_rate: 1.0000e-04
Epoch 8/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 46s 32ms/step - accuracy: 0.3687 - accuracy_a
t_five: 0.8888 - accuracy_at_three: 0.7183 - loss: 1.6683 - val_accuracy: 0.
3765 - val_accuracy_at_five: 0.8973 - val_accuracy_at_three: 0.7274 - val_lo
ss: 1.6283 - learning_rate: 1.0000e-04
Epoch 9/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 50s 35ms/step - accuracy: 0.3694 - accuracy_a
t_five: 0.8923 - accuracy_at_three: 0.7223 - loss: 1.6581 - val_accuracy: 0.
3758 - val_accuracy_at_five: 0.8978 - val_accuracy_at_three: 0.7298 - val_lo
ss: 1.6204 - learning_rate: 1.0000e-04
Epoch 10/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 87s 38ms/step - accuracy: 0.3726 - accuracy_a
t_five: 0.8943 - accuracy_at_three: 0.7258 - loss: 1.6451 - val_accuracy: 0.
3774 - val_accuracy_at_five: 0.8992 - val_accuracy_at_three: 0.7301 - val_lo
ss: 1.6162 - learning_rate: 1.0000e-04
Epoch 11/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 54s 38ms/step - accuracy: 0.3803 - accuracy_a
t_five: 0.8995 - accuracy_at_three: 0.7325 - loss: 1.6282 - val_accuracy: 0.
3794 - val_accuracy_at_five: 0.8995 - val_accuracy_at_three: 0.7324 - val_lo
ss: 1.6164 - learning_rate: 1.0000e-04
Epoch 12/70
```

```
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 55s 39ms/step - accuracy: 0.3750 - accuracy_a
t_five: 0.8998 - accuracy_at_three: 0.7343 - loss: 1.6256 - val_accuracy: 0.
3789 - val_accuracy_at_five: 0.9030 - val_accuracy_at_three: 0.7369 - val_lo
ss: 1.6069 - learning_rate: 1.0000e-04
Epoch 13/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 76s 35ms/step - accuracy: 0.3835 - accuracy_a
t_five: 0.9032 - accuracy_at_three: 0.7367 - loss: 1.6121 - val_accuracy: 0.
3817 - val_accuracy_at_five: 0.9023 - val_accuracy_at_three: 0.7365 - val_lo
ss: 1.6037 - learning_rate: 1.0000e-04
Epoch 14/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 50s 35ms/step - accuracy: 0.3840 - accuracy_a
t_five: 0.9051 - accuracy_at_three: 0.7396 - loss: 1.6098 - val_accuracy: 0.
3826 - val_accuracy_at_five: 0.9031 - val_accuracy_at_three: 0.7391 - val_lo
ss: 1.6027 - learning_rate: 1.0000e-04
Epoch 15/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 61s 42ms/step - accuracy: 0.3883 - accuracy_a
t_five: 0.9061 - accuracy_at_three: 0.7434 - loss: 1.5957 - val_accuracy: 0.
3820 - val_accuracy_at_five: 0.9017 - val_accuracy_at_three: 0.7371 - val_lo
ss: 1.6074 - learning_rate: 1.0000e-04
Epoch 16/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 71s 34ms/step - accuracy: 0.3881 - accuracy_a
t_five: 0.9093 - accuracy_at_three: 0.7454 - loss: 1.5915 - val_accuracy: 0.
3861 - val_accuracy_at_five: 0.9050 - val_accuracy_at_three: 0.7407 - val_lo
ss: 1.5975 - learning_rate: 1.0000e-04
Epoch 17/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 86s 37ms/step - accuracy: 0.3907 - accuracy_a
t_five: 0.9088 - accuracy_at_three: 0.7467 - loss: 1.5859 - val_accuracy: 0.
3833 - val_accuracy_at_five: 0.9055 - val_accuracy_at_three: 0.7420 - val_lo
ss: 1.5955 - learning_rate: 1.0000e-04
Epoch 18/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 88s 42ms/step - accuracy: 0.3860 - accuracy_a
t_five: 0.9113 - accuracy_at_three: 0.7479 - loss: 1.5823 - val_accuracy: 0.
3840 - val_accuracy_at_five: 0.9054 - val_accuracy_at_three: 0.7408 - val_lo
ss: 1.5945 - learning_rate: 1.0000e-04
Epoch 19/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 79s 40ms/step - accuracy: 0.3905 - accuracy_a
t_five: 0.9101 - accuracy_at_three: 0.7497 - loss: 1.5775 - val_accuracy: 0.
3809 - val_accuracy_at_five: 0.9043 - val_accuracy_at_three: 0.7376 - val_lo
ss: 1.6052 - learning_rate: 1.0000e-04
Epoch 20/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 58s 41ms/step - accuracy: 0.3953 - accuracy_a
t_five: 0.9146 - accuracy_at_three: 0.7534 - loss: 1.5635 - val_accuracy: 0.
3828 - val_accuracy_at_five: 0.9057 - val_accuracy_at_three: 0.7396 - val_lo
ss: 1.5971 - learning_rate: 1.0000e-04
Epoch 21/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 64s 28ms/step - accuracy: 0.3985 - accuracy_a
t_five: 0.9161 - accuracy_at_three: 0.7582 - loss: 1.5558 - val_accuracy: 0.
3817 - val_accuracy_at_five: 0.9045 - val_accuracy_at_three: 0.7379 - val_lo
ss: 1.6021 - learning_rate: 4.0000e-05
Epoch 22/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 43s 30ms/step - accuracy: 0.3989 - accuracy_a
t_five: 0.9168 - accuracy_at_three: 0.7597 - loss: 1.5523 - val_accuracy: 0.
3845 - val_accuracy_at_five: 0.9047 - val_accuracy_at_three: 0.7431 - val_lo
ss: 1.5961 - learning_rate: 4.0000e-05
Epoch 23/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 78s 27ms/step - accuracy: 0.3971 - accuracy_a
```

```
t_five: 0.9184 - accuracy_at_three: 0.7615 - loss: 1.5479 - val_accuracy: 0.
3848 - val_accuracy_at_five: 0.9063 - val_accuracy_at_three: 0.7445 - val_lo
ss: 1.5942 - learning_rate: 1.6000e-05
Epoch 24/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 43s 30ms/step - accuracy: 0.3996 - accuracy_a
t_five: 0.9186 - accuracy_at_three: 0.7599 - loss: 1.5462 - val_accuracy: 0.
3841 - val_accuracy_at_five: 0.9062 - val_accuracy_at_three: 0.7443 - val_lo
ss: 1.5927 - learning_rate: 1.6000e-05
Epoch 25/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 47s 33ms/step - accuracy: 0.3994 - accuracy_a
t_five: 0.9185 - accuracy_at_three: 0.7609 - loss: 1.5464 - val_accuracy: 0.
3859 - val_accuracy_at_five: 0.9064 - val_accuracy_at_three: 0.7442 - val_lo
ss: 1.5925 - learning_rate: 1.6000e-05
Epoch 26/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 73s 27ms/step - accuracy: 0.4013 - accuracy_a
t_five: 0.9183 - accuracy_at_three: 0.7588 - loss: 1.5490 - val_accuracy: 0.
3858 - val_accuracy_at_five: 0.9067 - val_accuracy_at_three: 0.7458 - val_lo
ss: 1.5950 - learning_rate: 1.6000e-05
Epoch 27/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 42s 29ms/step - accuracy: 0.3998 - accuracy_a
t_five: 0.9196 - accuracy_at_three: 0.7632 - loss: 1.5418 - val_accuracy: 0.
3843 - val_accuracy_at_five: 0.9065 - val_accuracy_at_three: 0.7451 - val_lo
ss: 1.5941 - learning_rate: 1.6000e-05
Epoch 28/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 40s 28ms/step - accuracy: 0.3989 - accuracy_a
t_five: 0.9207 - accuracy_at_three: 0.7612 - loss: 1.5443 - val_accuracy: 0.
3855 - val_accuracy_at_five: 0.9065 - val_accuracy_at_three: 0.7446 - val_lo
ss: 1.5930 - learning_rate: 6.4000e-06
Epoch 29/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 40s 27ms/step - accuracy: 0.4061 - accuracy_a
t_five: 0.9186 - accuracy_at_three: 0.7642 - loss: 1.5416 - val_accuracy: 0.
3860 - val_accuracy_at_five: 0.9059 - val_accuracy_at_three: 0.7443 - val_lo
ss: 1.5926 - learning_rate: 6.4000e-06
Epoch 30/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 42s 28ms/step - accuracy: 0.4032 - accuracy_a
t_five: 0.9192 - accuracy_at_three: 0.7658 - loss: 1.5418 - val_accuracy: 0.
3850 - val_accuracy_at_five: 0.9063 - val_accuracy_at_three: 0.7449 - val_lo
ss: 1.5933 - learning_rate: 2.5600e-06
Epoch 31/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 41s 29ms/step - accuracy: 0.4030 - accuracy_a
t_five: 0.9193 - accuracy_at_three: 0.7624 - loss: 1.5389 - val_accuracy: 0.
3845 - val_accuracy_at_five: 0.9063 - val_accuracy_at_three: 0.7451 - val_lo
ss: 1.5934 - learning_rate: 2.5600e-06
Epoch 32/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 45s 31ms/step - accuracy: 0.4022 - accuracy_a
t_five: 0.9185 - accuracy_at_three: 0.7627 - loss: 1.5399 - val_accuracy: 0.
3848 - val_accuracy_at_five: 0.9063 - val_accuracy_at_three: 0.7447 - val_lo
ss: 1.5933 - learning_rate: 1.0240e-06
Epoch 33/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 75s 27ms/step - accuracy: 0.4006 - accuracy_a
t_five: 0.9182 - accuracy_at_three: 0.7625 - loss: 1.5445 - val_accuracy: 0.
3848 - val_accuracy_at_five: 0.9062 - val_accuracy_at_three: 0.7448 - val_lo
ss: 1.5935 - learning_rate: 1.0240e-06
Epoch 34/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 39s 28ms/step - accuracy: 0.4022 - accuracy_a
t_five: 0.9197 - accuracy_at_three: 0.7634 - loss: 1.5398 - val_accuracy: 0.
```

```
3847 - val_accuracy_at_five: 0.9065 - val_accuracy_at_three: 0.7447 - val_lo
ss: 1.5933 - learning_rate: 1.0000e-06
Epoch 35/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 43s 29ms/step - accuracy: 0.4033 - accuracy_a
t_five: 0.9191 - accuracy_at_three: 0.7639 - loss: 1.5399 - val_accuracy: 0.
3850 - val_accuracy_at_five: 0.9062 - val_accuracy_at_three: 0.7446 - val_lo
ss: 1.5931 - learning_rate: 1.0000e-06
Epoch 36/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 79s 27ms/step - accuracy: 0.4019 - accuracy_a
t_five: 0.9190 - accuracy_at_three: 0.7653 - loss: 1.5410 - val_accuracy: 0.
3840 - val_accuracy_at_five: 0.9060 - val_accuracy_at_three: 0.7450 - val_lo
ss: 1.5935 - learning_rate: 1.0000e-06
Epoch 37/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 40s 28ms/step - accuracy: 0.3998 - accuracy_a
t_five: 0.9192 - accuracy_at_three: 0.7633 - loss: 1.5455 - val_accuracy: 0.
3848 - val_accuracy_at_five: 0.9063 - val_accuracy_at_three: 0.7449 - val_lo
ss: 1.5933 - learning_rate: 1.0000e-06
Epoch 38/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 43s 29ms/step - accuracy: 0.4054 - accuracy_a
t_five: 0.9203 - accuracy_at_three: 0.7657 - loss: 1.5376 - val_accuracy: 0.
3846 - val_accuracy_at_five: 0.9061 - val_accuracy_at_three: 0.7445 - val_lo
ss: 1.5933 - learning_rate: 1.0000e-06
Epoch 39/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 43s 30ms/step - accuracy: 0.4025 - accuracy_a
t_five: 0.9192 - accuracy_at_three: 0.7636 - loss: 1.5412 - val_accuracy: 0.
3842 - val_accuracy_at_five: 0.9060 - val_accuracy_at_three: 0.7447 - val_lo
ss: 1.5933 - learning_rate: 1.0000e-06
Epoch 40/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 77s 27ms/step - accuracy: 0.4047 - accuracy_a
t_five: 0.9200 - accuracy_at_three: 0.7653 - loss: 1.5347 - val_accuracy: 0.
3845 - val_accuracy_at_five: 0.9062 - val_accuracy_at_three: 0.7450 - val_lo
ss: 1.5932 - learning_rate: 1.0000e-06
Epoch 41/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 43s 30ms/step - accuracy: 0.4021 - accuracy_a
t_five: 0.9200 - accuracy_at_three: 0.7606 - loss: 1.5413 - val_accuracy: 0.
3842 - val_accuracy_at_five: 0.9060 - val_accuracy_at_three: 0.7448 - val_lo
ss: 1.5936 - learning_rate: 1.0000e-06
Epoch 42/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 78s 27ms/step - accuracy: 0.4014 - accuracy_a
t_five: 0.9214 - accuracy_at_three: 0.7658 - loss: 1.5363 - val_accuracy: 0.
3842 - val_accuracy_at_five: 0.9061 - val_accuracy_at_three: 0.7445 - val_lo
ss: 1.5938 - learning_rate: 1.0000e-06
Epoch 43/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 40s 28ms/step - accuracy: 0.3974 - accuracy_a
t_five: 0.9193 - accuracy_at_three: 0.7663 - loss: 1.5393 - val_accuracy: 0.
3841 - val_accuracy_at_five: 0.9059 - val_accuracy_at_three: 0.7448 - val_lo
ss: 1.5934 - learning_rate: 1.0000e-06
Epoch 44/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 42s 29ms/step - accuracy: 0.4044 - accuracy_a
t_five: 0.9200 - accuracy_at_three: 0.7644 - loss: 1.5370 - val_accuracy: 0.
3847 - val_accuracy_at_five: 0.9063 - val_accuracy_at_three: 0.7451 - val_lo
ss: 1.5933 - learning_rate: 1.0000e-06
Epoch 45/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 78s 26ms/step - accuracy: 0.4003 - accuracy_a
t_five: 0.9196 - accuracy_at_three: 0.7622 - loss: 1.5405 - val_accuracy: 0.
3843 - val_accuracy_at_five: 0.9059 - val_accuracy_at_three: 0.7452 - val_lo
```

```
ss: 1.5933 - learning_rate: 1.0000e-06
Epoch 46/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 45s 29ms/step - accuracy: 0.4040 - accuracy_a
t_five: 0.9182 - accuracy_at_three: 0.7633 - loss: 1.5403 - val_accuracy: 0.
3842 - val_accuracy_at_five: 0.9059 - val_accuracy_at_three: 0.7446 - val_lo
ss: 1.5937 - learning_rate: 1.0000e-06
Epoch 47/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 79s 27ms/step - accuracy: 0.4000 - accuracy_a
t_five: 0.9197 - accuracy_at_three: 0.7625 - loss: 1.5401 - val_accuracy: 0.
3843 - val_accuracy_at_five: 0.9063 - val_accuracy_at_three: 0.7450 - val_lo
ss: 1.5934 - learning_rate: 1.0000e-06
Epoch 48/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 41s 29ms/step - accuracy: 0.4012 - accuracy_a
t_five: 0.9186 - accuracy_at_three: 0.7645 - loss: 1.5377 - val_accuracy: 0.
3845 - val_accuracy_at_five: 0.9061 - val_accuracy_at_three: 0.7450 - val_lo
ss: 1.5933 - learning_rate: 1.0000e-06
Epoch 49/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 42s 29ms/step - accuracy: 0.4048 - accuracy_a
t_five: 0.9182 - accuracy_at_three: 0.7635 - loss: 1.5399 - val_accuracy: 0.
3840 - val_accuracy_at_five: 0.9060 - val_accuracy_at_three: 0.7450 - val_lo
ss: 1.5935 - learning_rate: 1.0000e-06
Epoch 50/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 45s 31ms/step - accuracy: 0.4010 - accuracy_a
t_five: 0.9196 - accuracy_at_three: 0.7628 - loss: 1.5393 - val_accuracy: 0.
3843 - val_accuracy_at_five: 0.9061 - val_accuracy_at_three: 0.7453 - val_lo
ss: 1.5937 - learning_rate: 1.0000e-06
Epoch 51/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 42s 30ms/step - accuracy: 0.4014 - accuracy_a
t_five: 0.9190 - accuracy_at_three: 0.7628 - loss: 1.5446 - val_accuracy: 0.
3841 - val_accuracy_at_five: 0.9060 - val_accuracy_at_three: 0.7452 - val_lo
ss: 1.5936 - learning_rate: 1.0000e-06
Epoch 52/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 43s 30ms/step - accuracy: 0.4041 - accuracy_a
t_five: 0.9194 - accuracy_at_three: 0.7646 - loss: 1.5351 - val_accuracy: 0.
3846 - val_accuracy_at_five: 0.9060 - val_accuracy_at_three: 0.7448 - val_lo
ss: 1.5934 - learning_rate: 1.0000e-06
Epoch 53/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 73s 24ms/step - accuracy: 0.4023 - accuracy_a
t_five: 0.9194 - accuracy_at_three: 0.7613 - loss: 1.5465 - val_accuracy: 0.
3843 - val_accuracy_at_five: 0.9065 - val_accuracy_at_three: 0.7454 - val_lo
ss: 1.5934 - learning_rate: 1.0000e-06
Epoch 54/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 43s 25ms/step - accuracy: 0.4005 - accuracy_a
t_five: 0.9188 - accuracy_at_three: 0.7617 - loss: 1.5406 - val_accuracy: 0.
3848 - val_accuracy_at_five: 0.9065 - val_accuracy_at_three: 0.7451 - val_lo
ss: 1.5932 - learning_rate: 1.0000e-06
Epoch 55/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 35s 25ms/step - accuracy: 0.3999 - accuracy_a
t_five: 0.9196 - accuracy_at_three: 0.7649 - loss: 1.5417 - val_accuracy: 0.
3844 - val_accuracy_at_five: 0.9061 - val_accuracy_at_three: 0.7453 - val_lo
ss: 1.5934 - learning_rate: 1.0000e-06
Epoch 56/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 41s 25ms/step - accuracy: 0.4027 - accuracy_a
t_five: 0.9189 - accuracy_at_three: 0.7659 - loss: 1.5365 - val_accuracy: 0.
3843 - val_accuracy_at_five: 0.9063 - val_accuracy_at_three: 0.7456 - val_lo
ss: 1.5934 - learning_rate: 1.0000e-06
```

```
Epoch 57/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 42s 26ms/step - accuracy: 0.4004 - accuracy_a
t_five: 0.9180 - accuracy_at_three: 0.7613 - loss: 1.5442 - val_accuracy: 0.
3839 - val_accuracy_at_five: 0.9062 - val_accuracy_at_three: 0.7454 - val_lo
ss: 1.5935 - learning_rate: 1.0000e-06
Epoch 58/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 39s 25ms/step - accuracy: 0.4011 - accuracy_a
t_five: 0.9182 - accuracy_at_three: 0.7630 - loss: 1.5428 - val_accuracy: 0.
3837 - val_accuracy_at_five: 0.9057 - val_accuracy_at_three: 0.7449 - val_lo
ss: 1.5938 - learning_rate: 1.0000e-06
Epoch 59/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 35s 24ms/step - accuracy: 0.4007 - accuracy_a
t_five: 0.9194 - accuracy_at_three: 0.7618 - loss: 1.5389 - val_accuracy: 0.
3842 - val_accuracy_at_five: 0.9061 - val_accuracy_at_three: 0.7450 - val_lo
ss: 1.5934 - learning_rate: 1.0000e-06
Epoch 60/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 36s 25ms/step - accuracy: 0.4019 - accuracy_a
t_five: 0.9204 - accuracy_at_three: 0.7661 - loss: 1.5361 - val_accuracy: 0.
3842 - val_accuracy_at_five: 0.9057 - val_accuracy_at_three: 0.7446 - val_lo
ss: 1.5938 - learning_rate: 1.0000e-06
Epoch 61/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 35s 25ms/step - accuracy: 0.4023 - accuracy_a
t_five: 0.9199 - accuracy_at_three: 0.7635 - loss: 1.5396 - val_accuracy: 0.
3840 - val_accuracy_at_five: 0.9056 - val_accuracy_at_three: 0.7447 - val_lo
ss: 1.5937 - learning_rate: 1.0000e-06
Epoch 62/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 35s 24ms/step - accuracy: 0.4004 - accuracy_a
t_five: 0.9189 - accuracy_at_three: 0.7621 - loss: 1.5414 - val_accuracy: 0.
3839 - val_accuracy_at_five: 0.9057 - val_accuracy_at_three: 0.7447 - val_lo
ss: 1.5942 - learning_rate: 1.0000e-06
Epoch 63/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 42s 25ms/step - accuracy: 0.4026 - accuracy_a
t_five: 0.9188 - accuracy_at_three: 0.7611 - loss: 1.5428 - val_accuracy: 0.
3840 - val_accuracy_at_five: 0.9057 - val_accuracy_at_three: 0.7452 - val_lo
ss: 1.5936 - learning_rate: 1.0000e-06
Epoch 64/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 40s 24ms/step - accuracy: 0.4006 - accuracy_a
t_five: 0.9202 - accuracy_at_three: 0.7647 - loss: 1.5412 - val_accuracy: 0.
3838 - val_accuracy_at_five: 0.9064 - val_accuracy_at_three: 0.7449 - val_lo
ss: 1.5932 - learning_rate: 1.0000e-06
Epoch 65/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 35s 25ms/step - accuracy: 0.4025 - accuracy_a
t_five: 0.9201 - accuracy_at_three: 0.7644 - loss: 1.5414 - val_accuracy: 0.
3836 - val_accuracy_at_five: 0.9055 - val_accuracy_at_three: 0.7451 - val_lo
ss: 1.5938 - learning_rate: 1.0000e-06
Epoch 66/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 41s 24ms/step - accuracy: 0.4024 - accuracy_a
t_five: 0.9192 - accuracy_at_three: 0.7629 - loss: 1.5374 - val_accuracy: 0.
3840 - val_accuracy_at_five: 0.9056 - val_accuracy_at_three: 0.7449 - val_lo
ss: 1.5940 - learning_rate: 1.0000e-06
Epoch 67/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 35s 25ms/step - accuracy: 0.4016 - accuracy_a
t_five: 0.9190 - accuracy_at_three: 0.7620 - loss: 1.5401 - val_accuracy: 0.
3834 - val_accuracy_at_five: 0.9063 - val_accuracy_at_three: 0.7450 - val_lo
ss: 1.5936 - learning_rate: 1.0000e-06
Epoch 68/70
```

```
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 41s 24ms/step - accuracy: 0.4024 - accuracy_a
t_five: 0.9200 - accuracy_at_three: 0.7638 - loss: 1.5388 - val_accuracy: 0.
3836 - val_accuracy_at_five: 0.9057 - val_accuracy_at_three: 0.7449 - val_lo
ss: 1.5937 - learning_rate: 1.0000e-06
Epoch 69/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 35s 25ms/step - accuracy: 0.4025 - accuracy_a
t_five: 0.9190 - accuracy_at_three: 0.7645 - loss: 1.5344 - val_accuracy: 0.
3841 - val_accuracy_at_five: 0.9057 - val_accuracy_at_three: 0.7448 - val_lo
ss: 1.5938 - learning_rate: 1.0000e-06
Epoch 70/70
1431/1431 ━━━━━━━━━━━━━━━━━━━━ 35s 25ms/step - accuracy: 0.4029 - accuracy_a
t_five: 0.9200 - accuracy_at_three: 0.7650 - loss: 1.5389 - val_accuracy: 0.
3841 - val_accuracy_at_five: 0.9056 - val_accuracy_at_three: 0.7448 - val_lo
ss: 1.5938 - learning_rate: 1.0000e-06
```

Out[154…

```
<Functional name=functional_6, built=True>
```

Observations

- Slightly better performance when reducing the learning rate to 1e-4
- Reduce LR on plateau: only efficient on training validation, as the LR starts to reduce when the model is overfitting
- 1e-7 is too low: the loss is not decreasing

Conclusion: for this problem, a constant learning rate seems to be the most efficient method

Accuracies

In [32]:

```
best_convolution, train_predicted, train_y_true, train_y_pred, test_predicte
```

```
2024-09-03 09:52:00.277772: E external/local_xla/xla/stream_executor/cuda/cu
da_driver.cc:266] failed call to cuInit: CUDA_ERROR_UNKNOWN: unknown error
2024-09-03 09:52:00.277820: I external/local_xla/xla/stream_executor/cuda/cu
da_diagnostics.cc:135] retrieving CUDA diagnostic information for host: theo
vld
2024-09-03 09:52:00.277829: I external/local_xla/xla/stream_executor/cuda/cu
da_diagnostics.cc:142] hostname: theovld
2024-09-03 09:52:00.278344: I external/local_xla/xla/stream_executor/cuda/cu
da_diagnostics.cc:166] libcuda reported version is: 555.42.6
2024-09-03 09:52:00.278370: I external/local_xla/xla/stream_executor/cuda/cu
da_diagnostics.cc:170] kernel reported version is: 555.42.6
2024-09-03 09:52:00.278376: I external/local_xla/xla/stream_executor/cuda/cu
da_diagnostics.cc:249] kernel version seems to match DSO: 555.42.6
2024-09-03 09:52:00.868309: W external/local_tsl/tsl/framework/cpu_allocator
_impl.cc:83] Allocation of 68001120 exceeds 10% of free system memory.
WARNING: All log messages before absl::InitializeLog() is called are written
to STDERR
I0000 00:00:1725349921.365322   74054 service.cc:146] XLA service 0x7f4dc400
ed10 initialized for platform Host (this does not guarantee that XLA will be
used). Devices:
I0000 00:00:1725349921.365530   74054 service.cc:154]   StreamExecutor devic
e (0): Host, Default Version
2024-09-03 09:52:01.446883: I tensorflow/compiler/mlir/tensorflow/utils/dump
_mlir_util.cc:268] disabling MLIR crash reproducer, set env var `MLIR_CRASH_
REPRODUCER_DIRECTORY` to enable.
I0000 00:00:1725349922.221149   74054 device_compiler.h:188] Compiled cluste
r using XLA!  This line is logged at most once for the lifetime of the proce
ss.
```

In [33]:
```python
print_accuracies(best_convolution, test_y_true, test_y_pred)
```

```
Accuracy: 38.33%
Balanced Accuracy: 20.96%
Accuracy for Top3: 73.88%
Accuracy for Top5: 90.18%
```

Overall accuracy is the best so far, but balanced accuracy is worse than with class weights. We can add this last technique to further improve our model.

Explainability

We now try to explain the results of our model, by using Shap and visualisation techniques. These last work well with Convolutional Neural Network to plot the filters and get an insight of what's happening.

Features importance

In [43]:
```python
import shap

explainer = shap.GradientExplainer(best_convolution, [train_moves, train_fea
```

```
# Shape of shap_values
# - First element: (nb_samples, WIDTH, HEIGHT, CHANNELS, nb_outputs)
# - Second element: (nb_samples, nb_features, nb_labels)
shap_values = explainer.shap_values([test_moves[:4], test_features.values[:4
```

In [139…

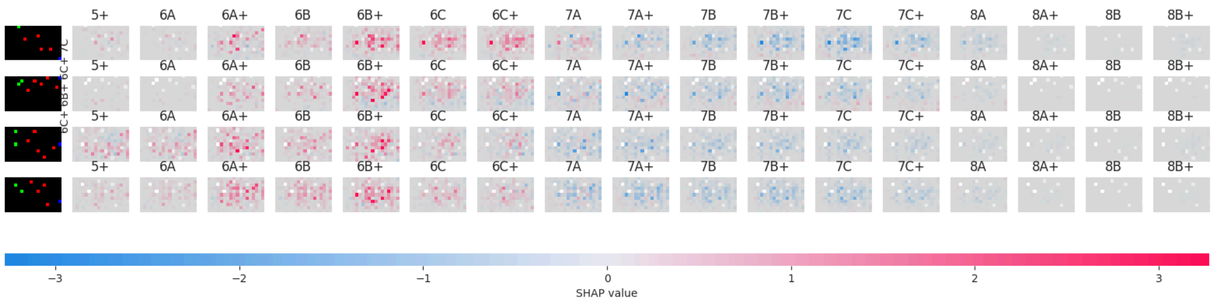
```
plt.figure(figsize=(60, 10))
shap.image_plot(
  [shap_values[0][:, :, :, :, i] for i in range(nb_labels)],
  test_moves[:4] * 255,
  labels=np.tile(all_grades, (nb_labels, 1)),
  show=False
)
plt.text(-350, -30, ' '.join(np.vectorize(lambda i: all_grades[i])(np.flip(t
```

```
<class 'NoneType'>
```

Out[139…

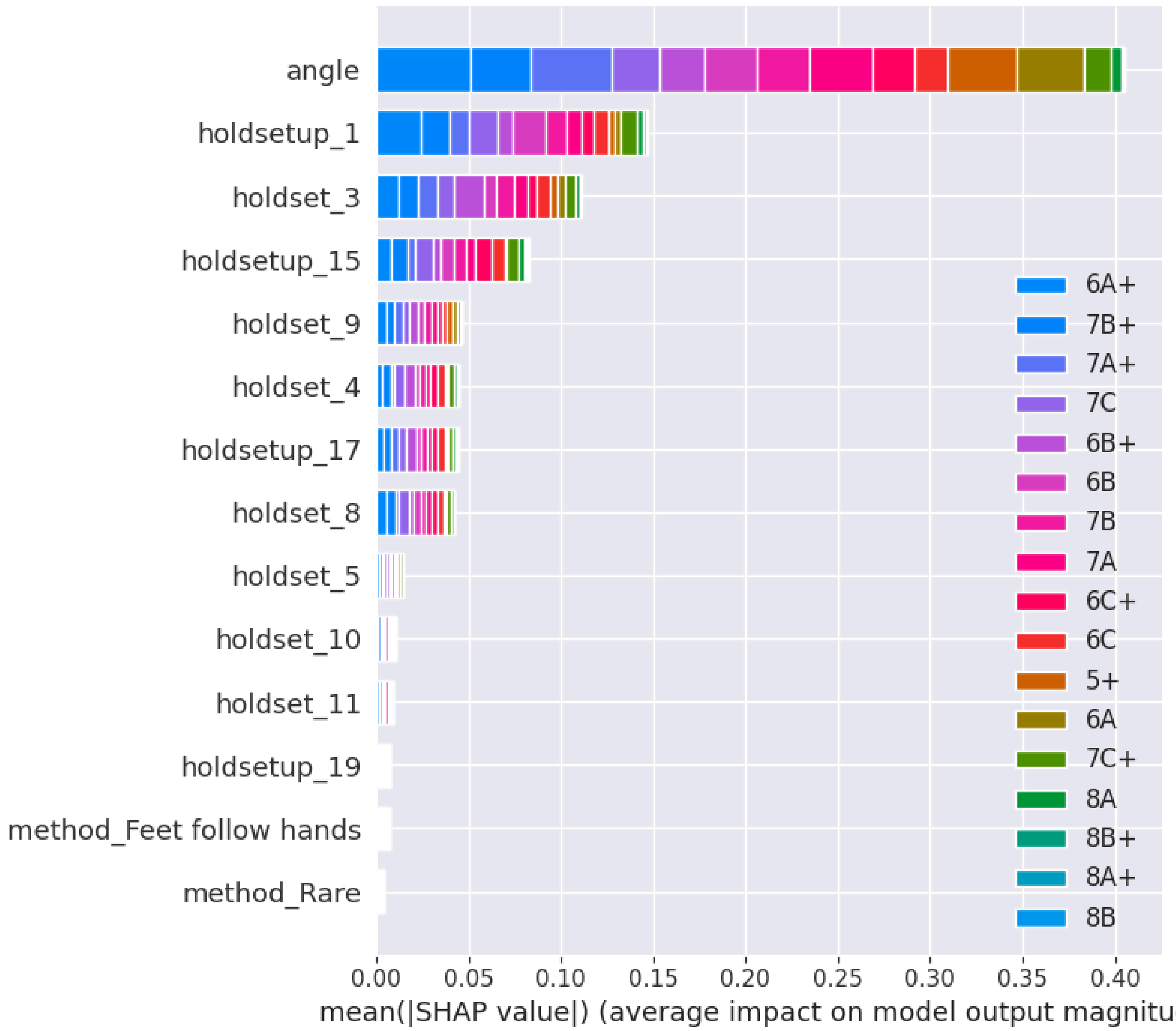
```
Text(-350, -30, '6C+ 6B+ 6C+ 7C')
```

```
<Figure size 6000x1000 with 0 Axes>
```



In [117…

```
shap.summary_plot(
  [shap_values[1][:, :, i] for i in range(nb_labels)],
  plot_type='bar',
  class_names=all_grades,
  feature_names=features.columns
)
```



- Board angle has the most influence on the grade, as it's more difficult to climb a steep route. It has almost an equal importance for all grades, if we take into account class imbalance
- Method has the least influence: not fully using feet doesn't make the route more difficult ; it's likely to be due to strong angles on moonboards (25° and 40°), thus the majority of the effort done by the climber is located in the arms and chest

Filters visualisation (TODO)

Conclusions

The conclusion will be written when the project has been completed, so after testing different configurations to reach the best performances, as well as explaining these results. Stay tuned!